Fault Pattern Recognition in Simulated Furnace Data

by

Carl Daniel Theunissen

Thesis presented in partial fulfilment
of the requirements for the Degree

of

MASTER OF ENGINEERING
(EXTRACTIVE METALLURGICAL ENGINEERING)

in the Faculty of Engineering
at Stellenbosch University

Supervisors
Doctor Tobias Louw
Professor Steven Bradshaw
Professor Lidia Auret

March 2021

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2021

PLAGIARISM DECLARATION

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. I agree that plagiarism is a punishable offence because it constitutes theft.

3. I also understand that direct translations are plagiarism.

4. Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.

5. I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

Student number: ……..

Initials and surname: ……..

Signature: ……..

Date: ……..

ABSTRACT

Modern submerged arc furnaces are plagued by blowbacks; hazardous occurrences where hot, toxic furnace freeboard gases are blown into the environment. While common occurrences, their causes are currently unknown, hence they cannot be predicted with mechanistic models. Data-driven models use data recorded from modern processes, like submerged arc furnaces, to recognize specific process conditions. This project aimed to identify and compare fault pattern recognition models that could be used for detecting and recognizing blowback-preceding conditions.

A simple submerged arc furnace model that emulates blowbacks was developed with which to generate large volumes of data for model comparison. This submerged arc furnace model was developed from mass- and energy balances over distinct furnace zones, and yielded a large dataset with dynamic- and nonlinear characteristics. This dataset contained observations from multiple distinct operating modes, and was deemed suitable for fault pattern recognition model evaluation.

A semi-supervised learning approach was selected as most suitable for recognizing blowback preceding conditions. Semi-supervised fault pattern recognition models are trained on a set of only blowback-preceding observations; this fits the typical constraints imposed by industrial datasets, where data is poorly defined and only a few observation of the target fault are labelled as such.

Principal component analysis (PCA), kernel PCA and input-reconstructing neural networks called auto-encoders are established semi-supervised pattern recognition methods. One-dimensional convolutional auto-encoders are neural network architectures that effectively compress multivariate time series, but their application to on-line fault pattern recognition is relatively novel. This work applied these methods to on-line fault pattern recognition for blowback prediction, and presented algorithms for applying these methods for semi-supervised fault pattern recognition tasks. Feature engineering has the largest impact on fault pattern recognition performance, therefore feature engineering techniques were applied as part of an overall approach to data-driven fault pattern recognition.

The investigation into the above fault pattern recognition models showed that kernel PCA's superior performance over standard PCA is limited to smaller datasets, and that large datasets must be compressed significantly before kernel PCA can be applied. Consequently this investigation found linear PCA to be superior to nonlinear kernel PCA for modelling large datasets. Both auto-encoders and the developed convolutional auto-encoders outperformed linear PCA modelling, highlighting the improved fault pattern recognition capabilities of nonlinear models.

This investigation found that one-dimensional convolutional auto-encoders were far more effective than the other presented models when applied to raw multivariate time series data, confirming that one-dimensional convolutional auto-encoders are effective at processing time series. However, the best performance was observed for auto-encoders models when applied to feature engineered data. This highlighted the guiding role that feature engineering should have in developing and implementing fault pattern recognition models.

ABSTRAK

Moderne onderdompelde boogoonde word gekwel deur terugploffings; gevaarlike gevalle waar warm, toksiese oondvryboordgasse in die omgewing geblaas word. Al is dit algemene verskynsels, is hul oorsake tans onbekend, en daarom kan hulle nie voorspel word deur meganistiese modelle nie. Datagedrewe modelle gebruik data wat opgeneem is van moderne prosesse, soos onderdompelde boogoonde, om spesifieke proseskondisies te herken. Hierdie projek het beoog om foutpatroonherkenningsmodelle te identifiseer en vergelyk om kondisies voor terugploffings op te spoor en te herken.

'n Eenvoudige onderdompelde boogoondmodel wat terugploffings naboots is ontwikkel waarmee groot volumes data vir modelvergelyking gegenereer kon word. Hierdie onderdompelde boogoondmodel is ontwikkel vanuit massa- en energiebalanse oor aparte oondsones, en het 'n groot datastel met dinamiese en nie-liniêre karakteristieke gelewer. Hierdie datastel het waarnemings van verskeie duidelike bedryfsmodus bevat, en is gepas geag vir foutpatroonherkenningsmodel se evaluasie.

'n Semi-toesighoudende leer benadering is gekies as mees gepas vir herkenning van terugploffings se voorafgaande kondisies. Semi-toesighoudende foutpatroonherkenningmodelle is opgelei uit 'n stel van slegs terugploffing-voorafgaande waarnemings; hierdie pas die tipiese beperkinge wat industriële datastelle oplê, waar data swak gedefinieer word en slegs 'n paar waarnemings van die teikenfout so benoem word.

Hoofkomponent analise (PCA), kern PCA en inset-rekonstrueering neurale netwerke wat outo-enkodeerders genoem word, is gevestigde semi-toesighoudende patroonherkenningsmetodes. Een-dimensionele konvolusionele outo-enkodeerders is 'n neurale netwerk argitektuur wat meervariaat tydreekse effektief kan kompres, maar hulle toepassing op op-lyn foutherkenning is relatief nuut. Hierdie werk het hierdie metodes op op-lyn foutpatroonherkenning vir terugploffing voorspelling toegepas, en algoritmes voorgestel om hierdie metodes vir semi-toesighoudende foutpatroonherkenningtake toe te pas. Kenmerkingenieurswese het die grootste impak op foutpatroonherkenning se doeltreffendheid, en daarom is kenmerkingenieurswesetegnieke gebruik as deel van 'n algehele benadering tot datagedrewe foutpatroonherkenning.

Die ondersoek in die bogenoemde foutpatroonherkenningmodelle het gewys dat kern PCA se superieure doeltreffendheid oor standaard PCA beperk is tot kleiner datastelle, en dat groot datastelle beduidend kompres moet word voordat kern PCA toegepas kan word. Vervolgens het hierdie ondersoek gevind dat liniêre PCA superieur is oor nie-liniêre kern PCA vir modellering van groot datastelle. Beide outo-enkodeerders en die ontwikkelde konvolusionele outo-enkodeerders het liniêre PCA-modellering oortref, wat die verbeterde foutpatroonherkenningkapasiteite van nie-liniêre modelle beklemtoon.

Hierdie ondersoek het gevind dat een-dimensionele konvolusionele outo-enkodeerders veel meer effektief is as die ander voorgestelde modelle wanneer dit toegepas word op rou meervariaat tydreeksdata, wat bevestig dat een-dimensionele konvolusionele outo-enkodeerders effektief is met prosessering van tydreekse. Die beste presteerder was egter waargeneem vir outo-enkodeerdermodelle toe dit op kenmerkingenieurswese toegepas is. Hierdie het die leidende rol wat kenmerkingenieurswese moet speel in ontwikkeling en implementering van foutpatroonherkenningmodelle, beklemtoon.

ACKNOWLEDGEMENTS

I wish to thank the following people in particular for their help in making this project a success:

- My parents, Danie and Izelle Theunissen, for their unwavering support and encouragement.
- Doctor Tobi Louw, for his guidance and open door.
- Professor Steven Bradshaw, for his contributions and supervision.
- Professor Lidia Auret, for her help and understanding.
- Amelia, for her patience and kindness.

TABLE OF CONTENTS

NOMENCLATURE

Symbols

A	Area	m^2
b	Kernel width	
C	Concentration	$mol \cdot m^{-3}$
c	Heat capacity	$J \cdot mol^{-1} \cdot K^{-1}$
F	Molar flow	$mol \cdot s^{-1}$
L	Height/thickness	m
M	Molar mass	$kg \cdot mol^{-1}$
N	Moles	mol
P	Pressure	Pa
p	Non-zero components	
Q	Heat transfer/generation	kW
R	Universal gas constant	$J \cdot mol^{-1} \cdot K^{-1}$
r	Reaction rate	$mol \cdot m^{-3} \cdot s^{-1}$
T	Temperature	K
V	Volume	m^3
v	Volume transfer	$m^3 \cdot s^{-1}$

Greek symbols

α	Confidence level-multiple values	
β	Confidence level-single value	
γ	Momentum factor	
δ	Specificity	
ε	Error	
η	Learning rate	
ρ	Density	$kg \cdot m^{-3}$
φ	Sensitivity	
ψ	Precision	

Acronyms

1D-CAE	One dimensional convolutional auto-encoder
AC	Alternating current
AE	Auto-encoder
ANN	Artificial neural network

AUC	Area under curve
CAE	Convolutional auto-encoder
CFD	Computational fluid dynamics
CNN	Convolutional neural network
CUSUM	Cumulative sum
EAF	Electric arc furnace
ECG	Electrocardiogram
EWMA	Exponentially weighted moving average
FPR	Fault pattern recognition
ML	Machine learning
MSPM	Multivariate statistical process monitoring
MTS	Multivariate time series
NOC	Normal operating conditions
OCC	One-class classifier
ODE	Ordinary differential equation
PC	Principal component
PCA	Principal component analysis
PGM	Platinum group metal
ROC	Receiver operating characteristic
SAF	Submerged arc furnace
SPM	Statistical process monitoring

1 INTRODUCTION

This chapter gives a brief overview of platinum group metal (PGM) production and introduces the concept of blowbacks in submerged arc furnaces. A brief overview of statistical pattern recognition in the context of blowback prediction is then provided. The importance of fault recognition in identifying blowback-preceding conditions is then highlighted as part the motivation of this study. The project aim and objectives are then provided, followed by the project scope. This chapter concludes by giving the layout of the thesis.

1.1 PGM production and blowbacks

South Africa hosts the majority of the world's PGM-reserves (Nell, 2004). These PGMs are found within the Bushveld Igneous Complex. Three ore types within the Bushveld Complex are exploited for their high PGM concentrations: the Merensky reef, the Plat reef and the UG2 reef (Cramer, 2001). PGMs are extracted by companies like Anglo American Platinum, Impala Platinum and Lonmin (Jones, 2005). PGM production is a dominating force in the South African mining sector, itself a cornerstone of the South African economy (Mudd, 2010).

The aforementioned companies extract PGMs from nickel-copper ores through a series of process steps. Each processing step increases the concentration of PGMs by reducing the bulk of the concentrate, or separates gangue from PGMs. Mined ore undergoes comminution, creating a sulphide concentrate. The sulphides are concentrated through flotation. Flotation concentrates are smelted and converted, yielding a PGM-rich copper-nickel matte. Hydrometallurgical treatments are used to separate base and precious metals. In the final step PGMs are refined into their pure forms (Jones, 2005).

The smelting step is critical to successfully extract PGMs from their ores (Nell, 2004). The concentration of PGMs increases tenfold in the smelting stage of the process (Jones, 2005). During smelting, submerged electrode arc furnaces melt dried concentrate into a copper-nickel sulphide matte that acts as a PGM collector (Nell, 2004). PGM smelting should be safe, effective and efficient (in that order) for the overall viability of the PGM extraction process.

A by-product of the smelting chemistry is the formation of sulphur dioxide and carbon monoxide gases from desulphurization and electrode oxidation reactions, respectively (Eksteen, 2011). Furthermore, smelting only occurs at high temperatures. These factors result in a furnace freeboard full of hazardous, hot gases. Blowbacks occur when the pressure in the furnace freeboard exceeds the pressure in the surrounding atmosphere. This causes hazardous furnace gases to escape from the furnace to the surrounding area, jeopardizing operator safety. A negative freeboard pressure is maintained by continuously extracting gases from the furnace, drawing in atmospheric air (Thethwayo, 2010). The air cools the furnace contents, consequently furnace efficiency is promoted by maintaining the pressure close to zero as possible.

Despite gas extraction, blowbacks are not uncommon in industrial submerged arc furnaces and their causes are unknown. A statistical pattern recognition solution to identify blowback-preceding conditions would provide a warning to operators of impending blowbacks, promoting safety, and allow freeboard pressure to be raised when blowbacks are not imminent, promoting efficiency.

1.2 Fault pattern recognition in blowback prediction

This thesis frames blowback prediction as a fault pattern recognition (FPR) problem; process faults that cause and precede blowbacks are expressed as characteristic patterns in process data, recognizing these patterns would warn submerged arc furnace operators about impending blowbacks. Submerged arc furnaces, like many modern chemical processes, are characterized by high product quality and energy efficiency demands, complex operations and large volumes of recorded historical data. These large data volumes recorded from submerged arc furnaces promote the use of statistical process monitoring for blowback prediction (Yin et al., 2010).

Modern processes require multivariate statistical process monitoring to detect and recognize process faults. Univariate statistical process monitoring approaches involve developing control charts for many individual process variables, overloading operators with data and obscuring useful information (Chen and Liao, 2002). Multivariate monitoring models combine process variables into single statistics to inform operators of process faults. Most multivariate monitoring models focus on modelling historical normal operating conditions and detect faults as deviations from normal conditions, but do not directly indicate which fault is occurring. Simpler faults can be recognized by isolating variables that contribute to faulty deviations, but complex faults require a comprehensive characterization of fault patterns (Westerhuis et al., 2000).

FPR models are multivariate monitoring models developed from historical data to detect and recognize specific faults (Hu et al., 2020). FPR models can only be developed if sufficient faulty observations are present in the historical dataset (Deng and Tian, 2013). In an ideal world, the data used to develop an FPR model would be completely characterized (meaning all faulty- and normal condition observations are labelled correctly). This would allow an FPR modelling approach where different process faults and normal are separated into distinct classes (Gredilla et al., 2013). Unfortunately, most real-world datasets used to develop FPR models are poorly characterized; only a few observations from one type of fault condition are labelled. This constraint has spurred the development of reconstruction-based one-class classifiers as FPR models (Villalba and Cunningham, 2007).

Reconstruction one-class classifiers are models trained to find effective, compressed representations of specific process faults (Tax, 2001). Fault patterns can be reconstructed from this representation with minimal error, while fault-free patterns will be reconstructed inaccurately. This facilitates FPR based on reconstruction error (Mazhelis, 2006). The various approaches to FPR are distinguished by how they find the compressed representations of process faults.

Principal component analysis (PCA) is an effective reconstruction-based one-class classifiers for FPR (Yin et al., 2010). PCA constructs the subspace of faulty process data containing significant linear correlations in fault patterns. Dynamic PCA (Ku et al., 1995) extends PCA to include significant autocorrelations of fault patterns in the PCA subspace. Fault patterns with characteristic linear correlations- and autocorrelations are effectively represented in the PCA subspace (Tax, 2001), but FPR performance deteriorates when fault patterns are characterized by nonlinear correlations.

Kernel PCA (Lee et al., 2004) is an efficient way of addressing the limitations of linear PCA. Kernel PCA employs the kernel trick to efficiently construct the subspace of faulty process data with significant nonlinear correlations in fault patterns, and is effective in recognizing nonlinear fault patterns in small datasets (Deng and Tian, 2013). However, kernel PCA models can only be developed on low rank approximations of large datasets (He and Zhang, 2018), leading to a drop in FPR performance when models are obtained from large volumes of training data, however the impact of approximation on monitoring performance has not been explored.

Auto-encoders (AEs) are neural networks that find the nonlinear subspace that accurately represent network inputs, then reconstruct inputs as the network outputs. They are therefore obvious candidates to use as reconstruction-based one-class classifiers (Tax, 2001). The earliest AE applied as a nonlinear alternative to PCA was a shallow feedforward network by Kramer (1991). Subsequent AEs applied in process monitoring generally increased in network depth to recognize more complicated process patterns (Hu et al., 2020). Unlike PCA, fault patterns characterized by nonlinearities are effectively represented in the AE subspace, and unlike kernel PCA, this subspace is obtainable from large volumes of training data.

Convolutional neural networks were developed for and completely outclass traditional feedforward networks in image processing applications (Ko and Kim, 2020). Convolutional neural networks extract simple, localized features from network inputs before moving on to more complicated features. This allows for more effective representations of network inputs across convolutional layers (Ismail Fawaz et al., 2019). Their adoption for statistical process monitoring has been slow, due to the intrinsic difference between images and multivariate time series, but the localized feature extraction of convolutional neural networks can lead to better input representation of multivariate time series. Recently, convolutional auto-encoders (CAEs) have been developed for compressing univariate signals (Wang et al., 2019) and for FPR on multivariate time series (Chen et al., 2020).

1.3 Project motivation

Submerged arc furnace blowbacks are hazardous events that jeopardize operator safety, and the large volumes of unlabelled data recorded on these processes favour reconstruction-based one-class classifiers for recognizing blowback-preceding conditions. Recognizing these conditions would promote operator safety by warning them of impending blowbacks and improve thermal efficiency by reducing unnecessary gas extraction. Furthermore, a statistical model that recognizes blowback-preceding conditions would be invaluable in identifying the root cause of blowbacks.

Although PCA, kernel PCA and AEs have been applied for FPR in multiple literature sources, in-depth comparisons of these techniques are rare. A comparison of these techniques in recognizing blowback-preceding conditions would inform future decision making when applying these techniques to industry data.

Kernel PCA is a well-established approach to FPR, and its performance is generally reported as superior to PCA in recognizing nonlinear fault patterns. However, most evaluations of kernel PCA have been

performed on small datasets, where low-rank approximations of training data are unnecessary. An evaluation of kernel PCA on a large dataset will shed light on the deterioration in FPR performance caused by training the model on an approximation of the process data.

CAEs are relatively novel approaches to FPR. Developing and applying a CAE model to recognizing blowback-preceding conditions would not only shed light on how such an FPR model can be applied in submerged arc furnaces in industry, but also on how this novel approach compares to the more established approaches like PCA, kernel PCA and AEs.

1.4 Project aim and objectives

The aim of this project is to contribute to safer submerged arc furnace operation by evaluating FPR models in recognizing blowback-preceding conditions in data that resembles industry data, and to contribute to the knowledge of FPR models by comparing different approaches to fault pattern reconstruction. The following objectives have been identified to achieve this aim:

1. Develop a simple furnace model that mechanistically simulates blowbacks. This model will be used to simulate data that resembles industry data w.r.t. dataset size. The model will facilitate model evaluation by providing a dataset that contains blowbacks, where the cause of the blowbacks are known.
2. Develop and implement statistical FPR models to recognize blowback-preceding conditions in the simulated furnace data. Specifically, PCA, kernel PCA, AEs and CAEs are implemented as reconstruction-based FPR algorithms.
3. Perform an objective evaluation of the FPR models' performance and compare their relative performance. A suitable evaluation metric that expresses each model's performance on the simulated data should be identified as part of this objective.

1.5 Project scope

1. A lumped parameter, dynamic furnace model that emulates blowbacks is developed to be used as a case study. This model should be expressed as a set of ordinary differential equations. The specific mechanism that causes blowbacks in this simulation will not be validated, due to the lack of knowledge on blowback causes. The goal of the furnace model is to generate data that plausibly mimic data obtained from industrial submerged arc furnaces.
2. An exploratory evaluation of reconstruction-based FPR models in identifying specific fault conditions is intended by this project. The scalability of these monitoring models to actual furnace data falls outside the scope of this thesis.
3. Monitoring models are evaluated on their ability to recognize blowback-preceding conditions, identifying the cause of blowbacks falls outside the scope of this project.

1.6 Thesis layout

Chapter 2 provides the reader with a detailed description on furnace operation in the context of the PGM production process. This chapter describes the series of steps in producing PGMs. The smelting step is described in detail, and the reaction chemistry within sulphide smelters are described. Blowbacks are

also discussed. This chapter also reviews the different approaches to simulating industrial furnaces, focusing on identifying which approach, or combination of approaches, would deliver a model that fits within the project scope.

Chapter 3 provides an overview of statistical methods in the context of statistical process monitoring, followed by a more detailed review of the reconstruction-based FPR approach used in this project. This chapter also reviews the application of PCA, kernel PCA, AEs and CAEs as FPR models. This chapter concludes by reviewing performance evaluation metrics of FPR models, with a focus on identifying evaluation metrics that objectively express FPR model performance in this project.

Chapter 4 provides the reader with a detailed description of the approach used and assumptions made to develop the furnace simulator used as a case study in this project. This chapter presents the approach used to emulating blowbacks in the developed furnace simulator. Finally, representative data generated by the developed furnace model are shown. This chapter effectively addresses the first objective identified for this project.

Chapter 5 presents the reader with the methodology used in implementing FPR models in recognizing blowback-preceding conditions in the simulated furnace data, addressing the second objective identified for this project. This chapter also presents the reader with the evaluation metric used to express model performance in this project.

Chapter 6 presents the performance of the implemented FPR models. This chapter then evaluates and discusses the observed model performance, highlighting significant findings obtained in the results. This chapter addresses the third objective identified for this project.

Chapter 7 concludes this thesis by summarizing the findings from this work, and presenting key insights gained. This chapter then provides recommendations for expanding the presented work.

2 PGM SMELTING AND FURNACE MODELLING REVIEW

This chapter provides a detailed description of the physical operation that this project is based on, and how this physical operation will be modelled. Section 2.1 informs the reader of the different ores exploited for PGM production. Section 2.2 gives an overview of the overall series of processing steps for converting concentrate to pure PGMs. Section 2.3 gives a description of the Polokwane smelter operated by Anglo Platinum, as well as the reaction chemistry within the smelter and a description of blowbacks. Finally, the different approaches in literature to modelling smelters are evaluated in section 2.4.

After reading this chapter, the reader should understand the sulphide smelting step in the context of PGM processing. The reader should be familiar with the conditions within the sulphide smelter and the phenomenon of blowbacks. The reader should also be familiar with how furnace modelling is approached in literature, and which approach is most suited for this project.

2.1 Ores exploited for PGMs

Most of the world's PGMs are produced from ore obtained from the Bushveld Igneous Complex. This complex hosts the largest reserves of PGMs in the world. It also contains base metals like nickel, copper and cobalt in quantities that are economically viable to recover (Cramer, 2001).

The Bushveld Complex primarily contains three ore types that are exploited for their PGM contents: the Merensky reef, the UG2 reef and the Plat reef. PGMs occur in the ores in conjunction with copper-nickel sulphides. The Plat- and Merensky reefs have similar mineralogical properties and are treated in the same manner (Nell, 2004). PGMs in the Plat- and Merensky reefs occur in a silicate substrate. The UG2 reef differs from the Plat- and Merensky reefs in its lower base metal sulphide contents and in its primary constituents: PGMs in the UG2 reef are found in a chromite matrix (Jones, 2005).

A blend of Merensky- and UG2 ore is typically processed for PGM production, with PGM concentrations ranging from 3 to 8 g/ton (Cramer, 2001). The ratio of Merensky ore is decreasing in favour of UG2-ore; this has caused submerged arc furnaces (SAFs) to be operated at high energy intensities to prevent chromite formation (Nell, 2004).

2.2 PGM production

PGM production requires a series of steps, illustrated in Figure 2.1, to convert PGM-containing ore into separated PGMs. These steps include comminution, flotation, smelting, converting, leaching and finally PGM refining (Mainza et al., 2005), and are discussed in some detail in this section. The smelting step is discussed in greater detail in the next section due to its relevance to this project.

2.2.1 Comminution

A variety of methods are employed for ore comminution. These include crushing-, ball-, rod- and autogenous milling. Normally, ores are milled to a classification of 60 % passing 74 µm. However, higher PGM prices may lead to milling circuits being run at higher capacities at the cost of recovery (Cramer, 2001). Particles that do not pass the classification size are recycled for further comminution. Hydrocyclones are employed for classification.

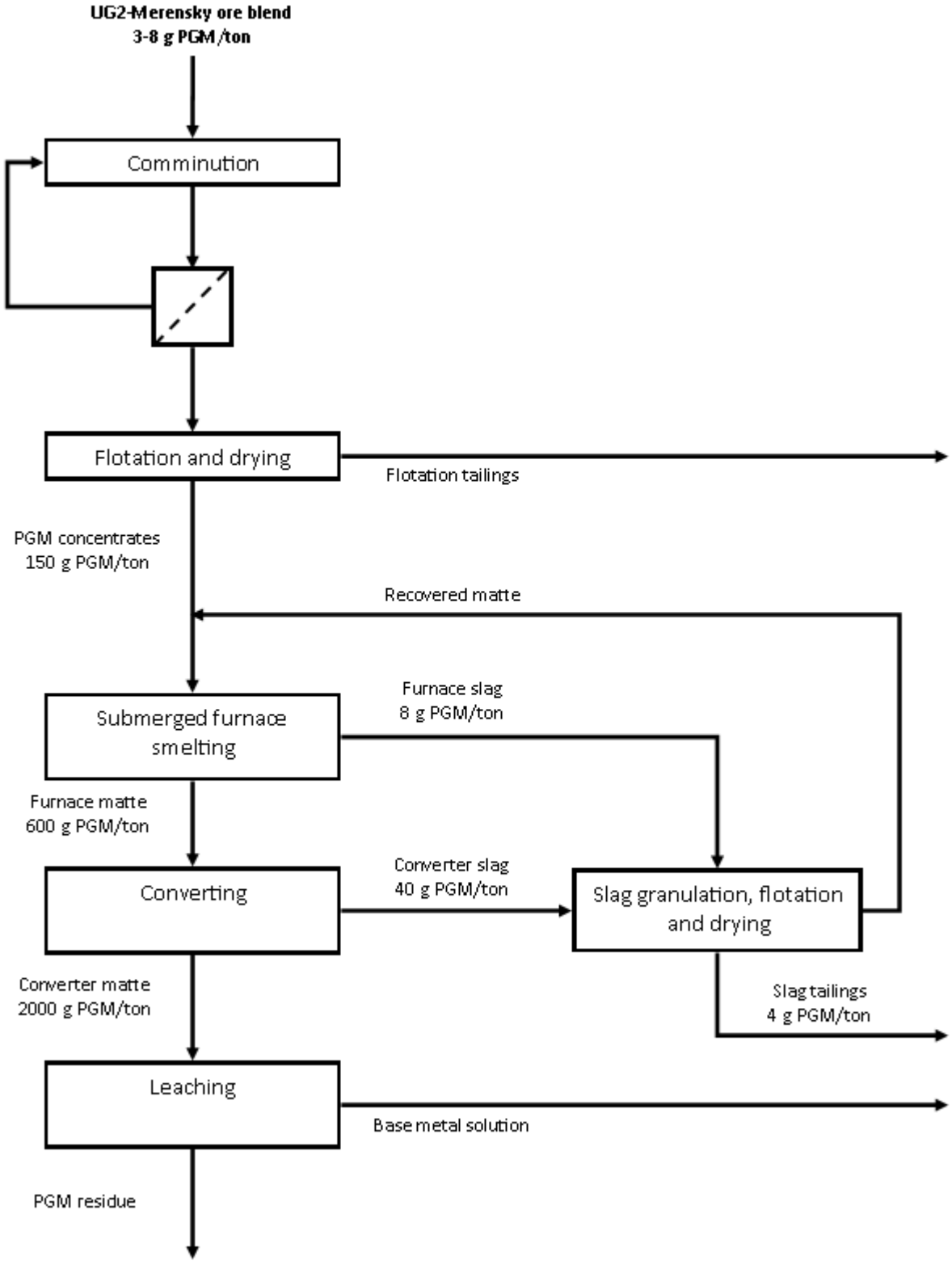

Figure 2.1: Process flow diagram of PGM production. By-products, like off-gases produced in the SAF-smelting and converting steps, are omitted, as well as recycle streams that do not directly influence the overall PGM recovery of the process.

2.2.2 Flotation

The concentrate from the comminution circuit is sent to a flotation circuit. The concentration of PGMs increases 30-fold in the flotation circuit, to a range between 100 to 400 g/ton (Jones, 2005). The flotation circuit separates gangue from valuable minerals using various collectors, activators, frothers and depressants (Cramer, 2001). Xanthates are the most common collectors. Copper sulphate is used as an activator for slower floating minerals. Depressant and frother use varies greatly depending on gangue mineralogy.

Wet concentrate from the flotation circuit is dried using a spray drier or a flash drier. This lowers the smelting energy requirements as well as decreasing blowback occurrence by preventing decomposable water molecules from entering the furnace.

2.2.3 Smelting

Smelting is discussed in greater detail in the next section. During smelting concentrate fed to the furnace is melted with electrical energy. The molten concentrate separates under gravity into two immiscible molten layers. The less dense slag layer contains silicates and oxides and very little PGMs. The denser matte layer contains sulphides and most of the PGMs in the feed (Crundwell et al., 2011a).

2.2.4 Converting

Matte from the furnace is transferred to a converting circuit. Here, iron sulphide (FeS) is oxidized to iron oxide (FeO) in Pierce-Smith converters, forming a second slag and matte phase. The concentration of PGMs in the converter matte increases through the removal of iron- and sulphur, yielding a metal-rich matte of base metals and PGM alloys (Nell, 2004).

2.2.5 Leaching

The converter matte is treated in a sulphuric acid leaching route. The base metals of the converted matte (nickel and copper) are soluble in sulphuric acid, while PGMs are not soluble. This results in a leach residue containing the PGMs of the converted matte (Jones, 2005).

2.3 Submerged arc smelting

The previous section described smelting in the context of PGM-processing. This section focuses on concepts related to submerged arc furnace design. This section also describes blowbacks.

2.3.1 Furnace layout and operation

PGM smelting in South Africa takes place exclusively in electric furnaces (Jones, 2005). Rectangular six-in-line submerged-arc electric furnaces are predominantly employed, and this furnace design will be considered further. Specifically, the layout of Anglo Platinum's Polokwane smelter will be discussed. This furnace's inner dimensions are 29.2 m long and 10.1 m wide (Van Manen, 2009). Six electrodes, each with a diameter of 1.6 m, are arranged in a line.

The smelter is of the Hatch design, and it is the largest capacity furnace in the platinum industry, rated at 68 MW (Hundermark et al., 2006). The furnace treats approximately 650 000 tonnes of concentrate per year, at an operating factor of 90%. Dried concentrate is charged pneumatically into the furnace using two feed bins above the furnace (Jones, 2005). A fluxing agent, like lime, is sometimes added with the concentrate (Nell, 2004) to assist with separating gangue from metallic sulphides.

During smelting, the concentrate melts into two liquid phases: a silicate- and oxide-rich slag with a density ranging from 2700 to 3300 kg/m^3 and a heavier sulphide matte with a density ranging from 4800 to 5300 kg/m^3. The liquid matte contains copper-, iron- and nickel sulphides, and the majority of the PGMs charged to the furnace (Jones, 2005). The two phases are tapped separately, from opposite ends, from the furnace.

A layer of unsmelted concentrate is maintained above the liquid slag layer. This unsmelted concentrate is called a 'black top' (Jones, 2005). This layer limits radiative heat transfer from the slag surface to the furnace walls and roof. The area above the concentrate layer is called the freeboard zone.

Off-gas is continuously withdrawn from the furnace. The furnace draught is controlled at -20 Pa gauge pressure. Off-gas temperatures typically range from 500˚C to 700˚C (Hundermark et al., 2006). The exhaust contains SO_2 from reactions of sulphide minerals (Jones, 2005).

The furnace is constructed from refractory materials and fitted with copper waffle coolers (Van Manen, 2009). The furnace hearth is constructed from mag-chrome refractory bricks. The upper sidewall is constructed from magnesite bricks and plate coolers. A high-alumina roof covers the furnace. Slag and matte is tapped from the furnace through water-cooled copper inserts and brick-lined, water-cooled copper tapblocks (Hundermark et al., 2006). The furnace has three matte tapholes and three slag tapholes.

2.3.2 Heat generation

The energy required for smelting is transferred to the concentrate using Söderberg-type electrodes with each pair rated at 56 MVA, with an applied current frequency of 50 Hz. The rating of the overall furnace is 168 MVA. Heat generation in the furnace occurs through Joule heating of the slag phase (Hundermark et al., 2006), and is sufficient for oxygen lancing and fuel injection to be unnecessary.

The electrodes are submerged in the slag phase beneath the concentrate 'black top'. The electrodes are continuously oxidized through reactions with oxides in the slag phase, releasing carbon monoxide (Sheng et al., 1998a). For every ton of concentrate fed, approximately 3 kg of electrode oxidizes (Crundwell et al., 2011a).

2.3.3 Smelting

PGMs are resistant to oxidation. This can be observed in nature, as PGMs occur in nature frequently as sulphides, such as cooperate and braggite (Cramer, 2001; Crundwell et al., 2011a) or alloys, like isoferroplatinum, while oxide minerals are relatively rare. This property plays an important role in smelting; PGMs do not form oxides, and therefore do not have an affinity for the oxide-rich slag phase.

Liquid matte is formed in the concentrate bed above the slag phase. After the base metal sulphides and PGMs have entered the concentrate bed, they start to convert to matte components through desulphurisation at temperatures around 650˚C (Eksteen, 2011). Matte drains from the concentrate bed before the chromite- and silicate portion of the bed starts to melt, due to the higher melting temperatures of these components (Crundwell et al., 2011a).

Base metal sulphides in the concentrate assist in PGM collection as sulphide droplets coalesce into the matte layer. PGM-concentrations are too low to form droplets with a size large enough to settle through the slag into the matte layer. They coalesce with sulphide droplets of base metals and both descend into the matte (Crundwell et al., 2011a). Concentrates fed to smelting furnaces are blended to contain sufficient sulphides to be effective PGM-collectors.

PGM smelting typically occurs at slag temperatures around 1350˚C, however smelting of UG2 concentrates requires higher temperatures around 1600˚C. Higher temperatures are required for UG2 concentrates due to the higher concentration of chromite in these concentrates. Chromite in the furnace feed results in highly refractory chromite spinel building up in the furnace, reducing the volume of the furnace. Chromite can consolidate into a third layer between matte and slag phases, preventing efficient slag and matte separation (Ritchie and Eksteen, 2011). At higher temperatures, chromite dissolves in the slag phase, preventing spinel build up (Jones, 2005; Nell, 2004). Reductive operating conditions within the furnace also inhibits chromite spinel formation (Thethwayo, 2010).

2.3.4 Furnace Blowbacks

The furnace freeboard contains hazardous gases from concentrate desulphurization- and electrode oxidation reactions. Gases are expelled whenever the furnace freeboard pressure exceeds the surrounding pressure. Furthermore, the off-gases extracted from the freeboard have temperatures in the region of 700 °C (Crundwell et al., 2011b).

The causes of furnace blowbacks are unknown at the present level of understanding, but they are avoided by extracting gas from the freeboard (Thethwayo, 2010). However, the cooling effect of air drawn into the furnace by excessively negative freeboard pressure is detrimental to the thermal efficiency of SAFs. This detrimental effect is pronounced when furnaces should operate at particularly high temperatures to avoid chromite spinel formation.

2.4 PGM furnace modelling review

This section addresses the first objective of this project: modelling a sulphide smelting submerged arc furnace. First, the model requirements to meet this objective are introduced. Next, the numerous furnace modelling approaches reported in literature are discussed. While many of the presented models cannot be applied in this project directly, the insights they give on heat generation, mass transfer, reaction chemistry and temperature distribution throughout submerged arc furnace baths will help in developing a model for this project. Finally, the reviewed models are summarized with a specific focus on how well they align with the model requirements for this project.

2.4.1 Model requirements

A simple submerged arc furnace model that mechanistically emulates furnace blowbacks is one of the objectives of this project. This model should provide a dataset wherein blowbacks occur, and where observations containing blowback-preceding conditions are known. This section will expand on the requirements of such a model to meet the stated project objectives. The model required by this project should:

1. Simulate sulphide-smelting submerged arc furnace behaviour.
2. Dynamically simulate furnace conditions, allowing blowbacks to be emulated.
3. Be computationally simple enough to generate large datasets corresponding to weeks of simulated furnace operation. A model expressed as a set of ordinary differential equations (ODEs) obtained from mass- and energy balances over distinct furnace zones would satisfy this requirement.
4. Approximate the matte, slag, concentrate- and freeboard furnace zones.
5. Mechanistically emulate furnace blowbacks.
6. Generate plausible variable values; the goal of this thesis is not to model submerged arc furnace behaviour, therefore the model is only required to generate variable profiles within the realm of plausibility.

2.4.2 Previous model formulations

The sulphide smelting submerged arc furnace simulators presented by Sheng et al. (1998a, 1998b), Bezuidenhout et al. (2009), Pan et al. (2011), Ritchie and Eksteen (2011) and Eksteen (2011) were considered to develop a model for this project. The dynamic steelmaking electric arc furnace simulators developed by Bekker et al. (2000), MacRosty and Swartz (2005) and Logar et al. (2012a, 2012b) aligned more closely to the stated model requirements than the submerged arc furnace models.

The submerged arc furnace simulators presented by Sheng et al. (1998a, 1998b) and Pan et al. (2011) only sought steady state model solutions, and are therefore unable to model dynamic furnace blowbacks. Likewise, the model presented by Eksteen (2011) focuses solely on matte droplet temperatures and settling rate and excludes dynamic furnace behaviour. The simulators presented by Bezuidenhout et al. (2009) and Ritchie and Eksteen (2011) are computational fluid dynamics (CFD) models. Simulated data generated by a CFD model of the submerged arc furnace would meet, and exceed, the scope of the project, but the CFD models reported in literature only approximate portions of the submerged arc furnace (by notably excluding the freeboard where blowbacks occur) and have prohibitively high computation costs. Using these CFD models to generate large data volumes corresponding to weeks of operation is infeasible.

The steelmaking electric arc furnace (EAF) models by Bekker et al. (2000), MacRosty and Swartz (2005) and Logar et al. (2012a, 2012b) aligned closely with the stated model requirements by simulating dynamic furnace freeboards and being simple enough to generate large quantities of data. Each of these models are presented as a set of ODEs obtained through mass- and energy balances over different furnace zones.

2.4.3 Model geometry

This section discusses how the models introduced in section 2.4.2 approximated submerged arc furnace geometry to facilitate modelling. These geometric approximations will guide which distinct furnace zones have to be considered separately when performing mass- and energy balances for developing ODEs.

The steady state furnace model developed by Sheng et al. (1998a, 1998b) approximated the furnace interior as a slag-, matte- and concentrate zone. The CFD model developed by Bezuidenhout et al. (2009) approximated the furnace interior using the same zones, but included furnace cooling units due to its significant effect on zone temperatures.

The steady state furnace model developed by Pan et al. (2011) was performed through separate mass- and energy balances for the matte-, slag-, concentrate- and freeboard zones. The model highlighted the importance of distinguishing between bulk concentrate zones and smelting zones.

2.4.4 Reaction heats and heat generation

The model developed by Sheng et al. (1998a, 1998b) found that heat generation occurs primarily in the slag zone through Joule heating and electrode arcing, and that bulk slag resistance decreases with increasing electrode immersion. The CFD models by Bezuidenhout et al. (2009) and Ritchie and Eksteen (2011) approximated all heat generation within the slag zone, and the model by Pan et al. (2011) continued this trend by assuming all heat generation is caused by Joule heating in the slag zone.

The high frequency of the AC current applied to the furnace electrodes has a negligible effect on heat generation (Bezuidenhout et al., 2009; Ritchie and Eksteen, 2011). The CFD model developed by Bezuidenhout et al. (2009) therefore replaced the 50 Hz applied current with 0.0167 Hz. The CFD model by Ritchie and Eksteen (2011) replaced the applied AC current with a time-averaged field.

The SAF model by Sheng et al. (1998a, 1998b) found that smelting reactions consume the overwhelming majority of energy supplied to submerged arc furnaces, with zone heating being a distant second. The heat consumed/provided by electrode oxidation, desulphurization or bath reactions were found to be negligible. Pan et al. (2011) only considered the enthalpy of smelting reactions in the concentrate zone when performing energy balances. Likewise, the CFD model by Bezuidenhout et al. (2009) omitted heat sinks other than smelting reactions. The energy balance over submerged arc furnaces can therefore be well approximated by smelting reactions and zone heating.

2.4.5 Heat transfer and temperature distribution

The CFD model developed by Bezuidenhout et al. (2009) approximated all heat transfer between zones through convection. The model developed by Pan et al. (2011) also assumed heat transfer through convection across zone interfaces. Furthermore, newly formed matte- and slag droplets were assumed to be at thermal equilibrium with the zones they report to upon formation.

The submerged arc furnace models developed by Sheng et al. (1998a, 1998b), Bezuidenhout et al. (2009) and Ritchie and Eksteen (2011) all concluded that the temperature distributions in the slag zone are homogeneous, but that the matte zone temperatures are stratified. Temperature stratification complicates heat transfer between lumped parameter zones, and is omitted from the steelmaking EAF models presented by Bekker et al. (2000), MacRosty and Swartz (2005) and Logar et al. (2012a, 2012b). The EAF model by Bekker et al. (2000) lumped the temperatures of liquid slag- and matte zones together.

The model by Pan et al. (2011) assumed that off-gases formed in the furnace bath are at thermal equilibrium with the concentrate above the slag zone, and that the furnace freeboard has a homogeneous temperature.

2.4.6 Concentrate smelting and mass transfer

The most significant mass transfer mechanisms highlighted in the presented submerged arc furnace models are the formation and settling of matte- and slag droplets from the concentrate layer to the furnace bath. The model by Eksteen (2011) investigated the effect of furnace conditions on matte droplet settling rates, and found that once released from the concentrate, matte droplets settle rapidly to the matte zone.

Chromite formation can have a significant impact on mass transfer, as they prevent slag- and matte separation. This phenomena was simulated by the CFD model developed by Ritchie and Eksteen (2011), who found that chromite formation is prevented with sufficient electrode immersion.

2.4.7 Furnace freeboard modelling

The submerged arc furnace model by Pan et al. (2011) was the only one examined in this review that considered the furnace freeboard. While their model did not provide the desired dynamic solution to the furnace freeboard, it did highlight three ways heat is transferred to/from the furnace freeboard: hot off-gases from the concentrate zone, convection with the top of the concentrate zone and cold ingress air.

The EAF model by Bekker et al. (2000) used a mole balance over the furnace freeboard considering the air drawn into the freeboard, the off-gases from the furnace bath and the gas extracted from the furnace to calculate the freeboard pressure using the ideal gas law. The model achieved numerical stability by lumping the freeboard pressure, slag- and matte zone temperatures together. The electric arc furnace model by Logar et al. (2012a, 2012b) used a similar mole balance over the furnace freeboard as Bekker et al. (2000), but did not lump the freeboard temperature with any other zones.

2.4.8 Summary of reviewed model features

This section presents reviewed model features that are most relevant to satisfying the model requirements listed in section 2.4.1. Table 2.1 lists these desired features, and indicates which reviewed models contained those features. Note that the ability to simulate blowbacks is an obvious desired model feature for this project, but this feature is omitted from Table 2.1 because none of the reviewed models are able to generate blowback data.

Table 2.1: Reviewed model features

Reviewed model	Submerged arc furnace	Dynamic model	Low computational cost	Model matte, slag and concentrate	Modelled freeboard
Sheng et al. (1998a, 1998b)	✓	✗	✓	✓	✗
Pan et al. (2011)	✓	✗	✓	✓	✓
Eksteen (2011)	✓	✗	✓	✗	✗
Bezuidenhout et al. (2009)	✓	✓	✗	✗	✗
Ritchie and Eksteen (2011)	✓	✓	✗	✗	✗
Bekker et al. (2000)	✗	✓	✓	✗	✓
MacRosty and Swartz (2005)	✗	✓	✓	✓	✓
Logar et al. (2012a, 2012b)	✗	✓	✓	✓	✓

Table 2.1 shows that none of the reviewed submerged arc furnace models satisfy all the model requirements for this project. Of the reviewed submerged arc models, the model by Pan et al. (2011) satisfies most of the requirements. However, its inability to generate dynamic data is a disqualifying drawback.

The electric arc furnace model by Logar et al. (2012a, 2012b) may not simulate a sulphide smelting furnace model, but it is able to generate dynamic data, it has a low computational cost allowing weeks of simulated data to be generated, it models distinct liquid matte-, slag- and concentrate zones, and models the furnace freeboard. All these features are desired for this project, therefore the furnace modelling and derivation approach used in this project will be guided by the model presented by Logar et al. (2012a, 2012b).

3 FAULT RECOGNITION REVIEW

This chapter informs the reader of various approaches to fault recognition, with a specific focus on the approach used in this project. Section 3.1 provides a brief overview of approaches to process monitoring, and introduces FPR in the broader context of statistical process monitoring. Section 3.2 discusses machine learning in FPR, and the different learning approaches to FPR, and highlights the strengths and limitations of the approach used in this project.

Section 3.3 provides an overview of PCA as an FPR model, followed by a high level description of PCA computations to highlight its advantages and limitations to the reader. Sections 3.4, 3.5 and 3.6 have similar structures to section 3.3, and informs the reader of kernel PCA, auto-encoders and convolutional auto-encoders, respectively. Finally, section 3.7 describes how pattern recognition performance is quantified and highlights the approaches used to evaluate FPR models in this project.

3.1 Fault pattern recognition in process monitoring

Monitoring models are used for fault detection and recognition in modern processes. Effective fault detection and recognition are crucial to meeting the operational safety, product quality and energy efficiency demands of these processes (Kassidas et al., 1998; Palma et al., 2015). This literature review has identified mechanistic- and data-driven modelling as the two main approaches to obtaining monitoring models, and this section explores the differences between monitoring models.

Mechanistic modelling uses first principles to derive a mechanistic process model (Kassidas et al., 1998). This model is used to simulate future process behaviour from current observations, allowing faults to be detected before failures can occur. Unfortunately, these models are exceedingly difficult to develop for modern processes; this has resulted in data-driven approaches enjoying priority over mechanistic approaches (Ammiche et al., 2018).

Data-driven approaches exploit the large volumes of data recorded on modern processes to construct statistical models (Yin et al., 2010). The earliest statistical models developed for fault detection and recognition are univariate control charts; these charts attempted to detect and recognize faults by flagging abnormal fluctuations in single variables (Li and Jeng, 2010; MacGregor and Kourti, 1995). Faults are rarely expressed in single variables, this spurred the development of multivariate statistical models (Westerhuis et al., 2000).

The earliest multivariate statistical models were constructed on normal operating data. These models could detect faults as deviations from normal conditions, but could not recognize those faults. Contribution charts were employed to isolate the variables that caused the faulty deviation (MacGregor and Kourti, 1995; Westerhuis et al., 2000). Isolating faulty variables may be sufficient for detecting simple faults, but complex faults would elude any statistical model trained on NOC data (Deng and Tian, 2013).

Luckily the large volumes of recorded process data often contain enough fault data to model specific faults. Machine learning techniques can use this historical fault data to train FPR models to recognize characteristic fault patterns (Deng and Tian, 2013). FPR models outperform normal operating condition models combined with contribution charts in recognizing specific fault patterns (Hu et al., 2020).

3.2 Fault pattern recognition

Machine learning is an important aspect of developing FPR models. This section provides a brief overview of machine learning, a detailed discussion of one-class classifiers and discusses the role of feature engineering in the context of this project.

3.2.1 Machine learning in fault pattern recognition

An FPR model is a classification model that matches a set of predictor variables to a class label using a model function and model parameters (Villalba and Cunningham, 2007). Machine learning is employed to find optimal model parameters using historical process data so that the classification model recognizes process conditions with the target fault (Salfner et al., 2010).

During training, the model assigns class labels to observations in the historical dataset. A loss function is computed for each model classification, quantifying how well the model classifies each observation (Ng, 2017). The optimal model parameters for the given model function are selected by minimizing the loss function, this is formally stated in equation 3-1:

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\big(\mathcal{E}(f, \boldsymbol{\theta}, \mathbf{X}_0)\big) \qquad [\,3\text{-}1\,]$$

$\mathcal{E}(f, \boldsymbol{\theta}, \mathbf{X}_0)$ is the total loss function for the model function f, model parameters $\boldsymbol{\theta}$ and training dataset, $\mathbf{X}_0$. $\mathcal{E}$ does not adequately express model performance because the model is fitted to $\mathbf{X}_0$, and a low $\mathcal{E}$ may be the result of overfitting a complex model function (Ng, 2005). Regularization techniques counteract overfitting during training, but a truly objective performance evaluation requires that the resulting model be tested on a separate dataset, $\mathbf{X}_1$ (Kramer, 1991).

The learning approach most suited for FPR depends on the model development dataset, $\mathbf{X}$. If $\mathbf{X}$ contains ample observations from each possible process operating condition, and all observations of the target fault are labelled, then a supervised binary classifier trained to separate the labelled- and unlabelled observations yields an FPR model that recognizes the target fault condition (Merelli and Luck, 2004).

Real-life historical datasets are rarely this well-defined and future process conditions are, by definition, not recorded in them. A binary classifier trained on $\mathbf{X}$ will struggle in separating the target fault from non-faulty conditions that are not recorded in $\mathbf{X}$. A semi-supervised one-class classifier identifies prominent characteristics in observations belonging to a single historical fault (Villalba and Cunningham, 2007), and therefore does not require $\mathbf{X}$ to be completely defined. Only historical faulty observations are needed to develop a one-class classifier. Reconstruction-based one-class classifiers are discussed in the next section.

3.2.2 Reconstruction-based one-class classifiers

Reconstruction-based one-class classifiers assume a model of the data-generating fault condition. Model parameters are obtained by finding a subspace of the target fault class to compress and reconstruct faulty observations accurately (Shyu et al., 2003). An effective reconstruction-based one-class classifier will reconstruct target fault observations with greater accuracy than fault-free observations, facilitating FPR by monitoring the reconstruction error.

Equations 3-2 to 3-5 formally present the reconstruction-based one-class classifier algorithm. An observation, $\mathbf{x}_i$, is reconstructed using the model function f and model parameters $\boldsymbol{\theta}$, yielding the reconstruction, $\hat{\mathbf{x}}_i$:

$$\hat{\mathbf{x}}_i = f(\mathbf{x}_i, \boldsymbol{\theta}) \quad [3\text{-}2]$$

The magnitude of the reconstruction error, ε_R, is computed:

$$\varepsilon_R = \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \quad [3\text{-}3]$$

The classification discriminant is calculated as the inverse of ε_R:

$$u_i = \frac{1}{\varepsilon_R} \quad [3\text{-}4]$$

$\mathbf{x}_i$ is recognized as faulty (C_1) if the discriminant is larger than a recognition threshold, τ. Else, $\mathbf{x}_i$ is not recognized by the reconstruction one-class classifier:

$$\gamma_i = \begin{cases} C_1 \; if \; u_i \geq \tau \\ C_0 \; if \; u_i < \tau \end{cases} \quad [3\text{-}5]$$

Reconstruction-based one-class classifiers hold key advantages over other classifiers of this type; they directly model faulty process behaviour, unlike density-based one-class classifiers (Mazhelis, 2006). Furthermore, recognition thresholds applied to the reconstruction error are independent of the model; this allows simple optimization of the model by adjusting recognition thresholds, unlike boundary-based one-class classifiers (Tax, 2001).

Reconstruction-based methods are doubly susceptible to the overfitting phenomenon when applied to process data with many variables. This is because the model fits as many output variables as there are input variables, and feature engineering is crucial for reducing variance in the model development dataset. Furthermore, a theoretical basis for the reconstruction error recognition threshold, τ, does not exist, and should therefore be obtained empirically (Tax, 2001).

3.2.3 Feature engineering for online FPR

Feature engineering is the manual creation of features from a dataset that is more suitable for model development. Broadly speaking, features are engineered to meet the following objectives in the context of machine learning (James et al., 2012):

1. Create informative features that models do not have to learn themselves.
2. Reduce the size of the model development dataset by removing redundant features.
3. Improve model generalization by reducing variance in the model development dataset.

Feature engineering techniques require expert knowledge to find features that are most suited for model development. These techniques are less complicated than the automated machine learning techniques discussed in sections 3.3 to 3.6, but often have a greater impact on recognition performance than the choice of machine learning model (Salfner et al., 2010).

Data scaling is the most common form of feature engineering used in pattern recognition (Bishop, 2006). Datasets are usually standardized before machine learning algorithms are applied to them; standardization removes scale-dependent features that are irrelevant in pattern recognition applications.

Calculating statistics from a moving window of measurements is an established approach to feature engineering (Susto et al., 2018). Individually irrelevant variables from a signal can be incorporated together in a single value that expresses a more relevant feature (Jiang et al., 2018; Kubben et al., 2019).

Signal de-trending is another standard approach to engineering features from multivariate time series (Trovero and Leonard, 2018). A signal can be viewed as a composite of distinct signal components, with each signal representing signal variation over different time scales. Equation 3-6 states formally how a measured variable, x_i, is decomposed into its components through additive decomposition:

$$x_i = T_i + C_i + S_i + N_i \qquad [3\text{-}6]$$

Faulty conditions do not form a part of normal operation, and the fluctuations they cause will not be expressed in long-term signal trend (T_i) or cyclical (C_i) components of measurements. Removing these components from measurements is called de-trending, and allows model development on the relevant seasonal (S_i) component of signals (Carbone, 2009). Note that signal de-trending does not remove high variance noise components (N_i) from signals.

3.3 Principal component analysis

Principal component analysis (PCA) is the most common statistical modelling approach to feature learning (Charte et al., 2020; Zhang et al., 2018), and is a prominent data-driven model used for process monitoring. PCA is applied in process monitoring by finding the directions of significant linearly uncorrelated variance in recorded data of the modelled process condition (Singhal and Seborg, 2002). These directions (called principal components) constitute a linear subspace of target process conditions. Observations with similar correlation structures to the target process conditions are well-represented in this subspace and can be reconstructed accurately, therefore PCA is an ideal model to recognize process conditions characterized by distinct linear correlation structures (Mazhelis, 2006).

MacGregor and Kourti (1995) used PCA to model the normal condition data in batch- and continuous chemical processes. While that study did not explore fault recognition, it did demonstrate that PCA can recognize specific process conditions; faults were detected when the model did not recognize observations as normal. Misra et al. (2002) also modelled normal condition data using PCA to investigate fault detection on industrial boiler datasets. The study by Ku et al. (1995) showed that PCA models built on specific fault data from the Tennessee Eastman process simulation can distinguish that fault from other simulated process conditions using reconstruction.

3.3.1 PCA computations

PCA models a process by finding the principal components of historical process data, $\mathbf{X} \in \Re^{n \times m}$, then selecting significant components to construct the PCA subspace. Principal components are computed through eigenvalue decomposition of the historical dataset's covariance matrix (Wise et al., 1990). This is shown in equation 3-7:

$$\mathbf{X}^{\mathbf{T}}\mathbf{X}\mathbf{v}_j = \lambda_j \mathbf{v}_j \qquad [3\text{-}7]$$

PCA is scale-sensitive, therefore $\mathbf{X}$ is standardized before a principal component, $\mathbf{v}_j \in \Re^m$, is calculated. n and m are the number of observations and variables in $\mathbf{X}$, respectively. λ_j quantifies the variance in $\mathbf{X}$ captured on $\mathbf{v}_j$. The significance of $\mathbf{v}_j$ is expressed by the fraction of total variance captured on it (Wise et al., 1990). This fraction of variance is calculated with equation 3-8:

$$\eta_j = \frac{\lambda_j}{\sigma_{\mathbf{X}}^2} = \frac{\lambda_j}{\sum_{j=1}^{m} \lambda_j} \qquad [3\text{-}8]$$

$\sigma_{\mathbf{X}}^2$ is the total variance in $\mathbf{X}$. The PCA subspace, $\mathbf{V} \in \Re^{m \times v}$, only contains the most significant principal components; retaining insignificant components causes noise to be represented in the PCA subspace. Selecting the number of significant components, v, is therefore crucial to PCA modelling (Wise et al., 1990). Graphical approaches to selecting v, like the widely used Scree test (Ledesma et al., 2015), are inherently subjective, therefore v is best approached as a design parameter for PCA models.

The PCA subspace, $\mathbf{V}$, represents a global optimum of the PCA model (Tax, 2001); no loss function is calculated to update and optimize model parameters. This highlights a key strength of PCA: unlike other data-driven models, PCA model parameters are not calculated iteratively. These model parameters are also inherently regularized if fewer principal components are retained than the dimension of the modelled data. The reconstruction model function for PCA, f_{PCA}, is given by equation 3-9:

$$\hat{\mathbf{x}}_i = f_{PCA}(\mathbf{x}_i, \mathbf{V}) = \mathbf{x}_i \mathbf{V} \mathbf{V}^{\mathrm{T}} \qquad [3\text{-}9]$$

Figure 3.1 illustrates PCA-based pattern recognition. Most of the significant variance in the dataset is captured on the first component, and this component is selected as the model subspace. The reconstruction error expresses how well a data point is approximated by the subspace.

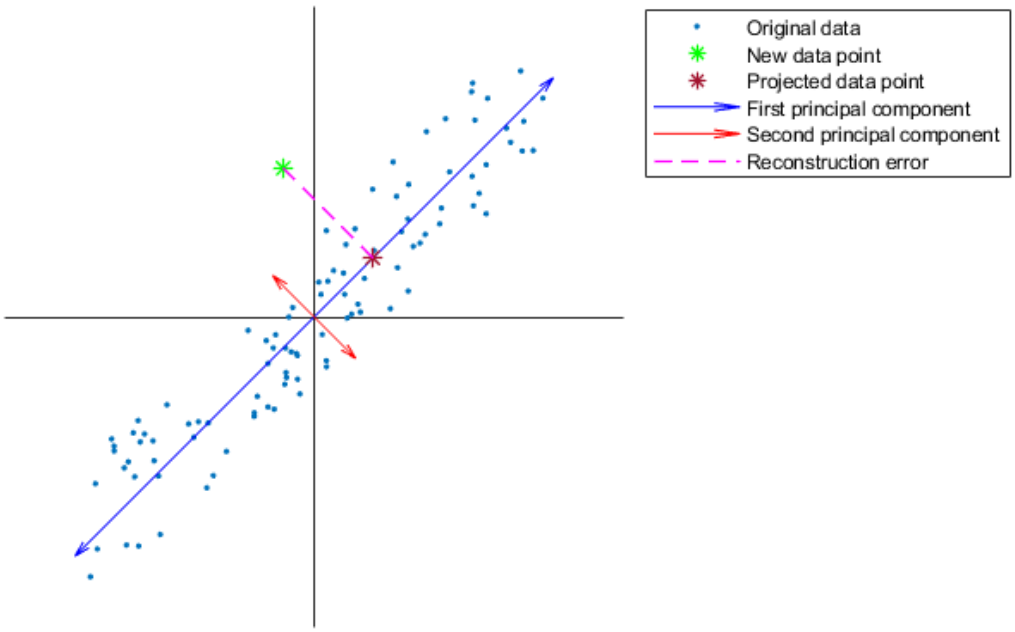

Figure 3.1: Illustration of PCA-data reconstruction. A new data point (green star) is projected onto the first principal component (blue arrow), yielding the projected data point (dark crimson star). The difference between the new data point and the projected data point is the reconstruction error.

3.3.2 Limitations of PCA

Linear correlations between variables are well-approximated in the PCA subspace, but the subspace excludes autocorrelations between observations. PCA is therefore only ideally suited to static process data (Dong and Qin, 2018; Misra et al., 2002). However, chemical processes like submerged arc furnaces are rarely at steady state and variables fluctuate according to intrinsic process dynamics. Monitoring algorithms that fail to address this time dependence would have limited performance.

Ku et al. (1995) presented a simple modification of PCA called dynamic PCA to address this limitation. Using dynamic PCA, observations (both new and historical) are lagged, incorporating previous values in each observation – this is shown in equation 3-10, where an observation, $\mathbf{x}_i$, is lagged l times. The dynamic PCA subspace constructed from lagged observations approximates linear autocorrelations. This dynamic PCA modification improved monitoring performance over standard PCA.

$$\mathbf{x}_i^L = \left[\mathbf{x}_i\ \mathbf{x}_{i-1}\ \mathbf{x}_{i-2} \cdots\ \mathbf{x}_{i-(l-2)}\ \mathbf{x}_{i-(l-1)}\ \mathbf{x}_{i-l}\right] \qquad [\ 3\text{-}10\ ]$$

PCA only identifies linear correlation structures, and characteristic nonlinearities may be discarded on insignificant components (Jain et al., 2000; Wise et al., 1990). Augmenting data by mapping observations to a nonlinear feature space exponentially increases computational requirements, a phenomenon called the curse of dimensionality (Khediri et al., 2011; Liu et al., 2011). Kernel PCA, discussed in the next section, modifies PCA to avoid this curse.

3.4 Kernel principal component analysis

Kernel PCA is a modification of PCA introduced by Schölkopf et al. (1998b) to find a nonlinear PCA subspace without explicitly mapping the model development dataset, $\mathbf{X}$. Since nonlinear features are mapped implicitly, kernel PCA provides the opportunity to construct subspaces using potentially infinite dimensional features. Observations with nonlinear correlation structures can therefore be well-approximated in the kernel PCA subspace, allowing fault patterns characterized by nonlinear correlation structures to be recognized. Kernel PCA modelling algorithms also yield a globally optimized model without iterative parameter learning (Ge et al., 2009).

Lee et al. (2004) applied kernel PCA in monitoring a simulated wastewater treatment plant, and found that kernel PCA offered improved performance over standard PCA. This paper also showed how reconstruction error is computed implicitly using kernel PCA. Choi and Lee (2004) expanded kernel PCA using the same time-lagged data extension as was used for regular PCA by Ku et al. (1995), and dynamic kernel PCA outperformed standard PCA in simulated process data.

Ge et al. (2009), Zhang et al. (2012) and Deng and Tian (2013) applied kernel PCA on simulated data from the Tennessee Eastman process for process monitoring. The approach used by Deng and Tian (2013) specifically focused on fault pattern recognition; observations were assigned to historical faults according to similarity.

3.4.1 Kernel PCA computations

Computing principal components from datasets mapped through nonlinear expansions is too computationally demanding to be done explicitly. If $\mathbf{Y} \in \Re^{n \times M}, M \gg m$, is the nonlinear mapping of $\mathbf{X} \in \Re^{n \times m}$, then equation 3-11 shows how components are computed explicitly:

$$\mathbf{Y^T Y v}_j = \lambda_j \mathbf{v}_j \qquad [\ 3\text{-}11\ ]$$

While the above eigenvalue decomposition cannot be computed for extensive nonlinear mapping, the components obtained from this feature space would still be linear combinations of the rows of $\mathbf{Y}$ (Lee et al., 2004), shown explicitly in equation 3-12:

$$\mathbf{Y^T a}_j = \mathbf{v}_j \qquad [\ 3\text{-}12\ ]$$

These linear combinations, $\mathbf{a}_j$, can be computed from the outer product of the mapped dataset in the same way that principal components are calculated from the inner product:

$$\mathbf{Y Y^T a}_j = \mathbf{K a}_j = \lambda_j \mathbf{a}_j \qquad [\ 3\text{-}13\ ]$$

The outer product of the mapped dataset, $\mathbf{Y Y^T}$, is called the kernel matrix, $\mathbf{K} \in \Re^{n \times n}$. Note that λ_j is still computed, allowing significant components to be identified even if they are not defined explicitly. Mercer kernels are functions that populate $\mathbf{K}$ implicitly from the unmapped dataset, $\mathbf{X}$. Gaussian kernel product functions are among the most widely used for kernel PCA (Zhang, 2008):

$$[\mathbf{K}]_{ij} = \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = e^{\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{b}\right)} \qquad [\ 3\text{-}14\ ]$$

Gaussian kernels map inputs into an infinite dimensional space (Shashua, 2009), allowing nonlinear feature extraction that would be impossible with standard PCA. The output of the Gaussian kernel expresses similarity between observations (Wang, 2012). Very dissimilar observations would populate the kernel matrix, $\mathbf{K}$, with zeros. Identical observations would populate the kernel matrix with ones. The kernel width, b, is a design parameter controlling this fit sensitivity, and its selection has a large effect on model performance (Keerthi and Lin, 2003). The optimal value of b is application-specific, and the approach used in this project to define this value is given as part of the methodology in section 5.6.

Avoiding explicit feature mapping prevents kernel PCA from reconstructing mapped feature inputs, and explicit reconstruction functions cannot be defined. Lee et al. (2004) showed that the reconstruction error can be calculated implicitly using equation 3-15 to facilitate process monitoring:

$$\varepsilon_{R,\,kPCA}(\mathbf{x}_i) = \mathbf{k}_i \mathbf{A_P} \mathbf{A_P}^{\mathrm{T}} \mathbf{k}_i^{\mathrm{T}} - \mathbf{k}_i \mathbf{A_V} \mathbf{A_V}^{\mathrm{T}} \mathbf{k}_i^{\mathrm{T}} \qquad [\ 3\text{-}15\ ]$$

Equation 3-15 assumes that the difference between the variance retained on all principal components with nonzero eigenvalues and the variance retained on significant principal components approximates the reconstruction error well. $\mathbf{A_P} \in \Re^{n \times p}$ is the matrix of linear component combinations, $\mathbf{a}_j$, that correspond to components with nonzero variance retention, and is computed with equation 3-16:

$$\mathbf{A_P} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p];\ p = \max(p)\ \text{s.t.}\ \lambda_p > 0 \qquad [\ 3\text{-}16\ ]$$

$\mathbf{A_V} \in \Re^{n \times v}$ is the matrix containing the linear combinations that correspond to the first v principal components. $\mathbf{A_V}$ represents the nonlinear kernel PCA subspace. $\mathbf{k}_i \in \Re^n$ is the kernel vector populated with the kernel product of a new observation, $\mathbf{x}_i$, and the original model development dataset, $\mathbf{X}$. Like standard PCA, Kernel PCA is scale sensitive; both $\mathbf{K}$ and $\mathbf{k}_i$ have to be centred before applying equations 3-13 or 3-15 (Schölkopf et al., 1998b):

$$\mathbf{K}_c = \mathbf{K} - \mathbf{UK} - \mathbf{KU} + \mathbf{UKU} \qquad [3\text{-}17]$$

$$\mathbf{k}_c = \mathbf{k}_i - \mathbf{uK} - \mathbf{k}_i\mathbf{U} + \mathbf{uKU} \qquad [3\text{-}18]$$

$\mathbf{U} \in \Re^{n \times n}$ $\left([\mathbf{U}]_{ij} = \frac{1}{n}\right)$ and $\mathbf{u} \in \Re^n$ $\left([\mathbf{u}]_i = \frac{1}{n}\right)$ are matrices used to centre $\mathbf{K}$- and $\mathbf{k}_i$ in the feature space, respectively.

3.4.2 Limitations of kernel PCA

Kernel PCA performs eigenvalue decomposition on the kernel matrix, $\mathbf{K}$. The size of this matrix grows exponentially with the number of samples in $\mathbf{X}$. This decomposition has a computation demand in the order of $O(n^3)$, meaning that the computational demands of kernel PCA grows cubically with larger datasets (He and Zhang, 2018; Zhang and Kwok, 2010).

This is not a problem in smaller datasets; nonlinear principal components can be computed implicitly using all available observations, but real-world datasets are large, rendering kernel PCA infeasible. Low rank approximations of $\mathbf{X}$ are therefore needed to apply kernel PCA to industrial data (Cheng et al., 2009). Unfortunately, no approximation of $\mathbf{K}$ will be as good as the original, because $\mathbf{K}$ has full rank (Schölkopf et al., 1998a).

k-means clustering is an established data partitioning technique that has been used to find low rank approximations of $\mathbf{K}$ (He and Zhang, 2018). The algorithm partitions data into k clusters, with each observation assigned to the cluster with the nearest cluster mean (Shashua, 2009). Figure 3.2 illustrates a dataset is partitioned into five clusters. The resulting cluster centroids can be used to compute a low rank approximation of $\mathbf{K}$.

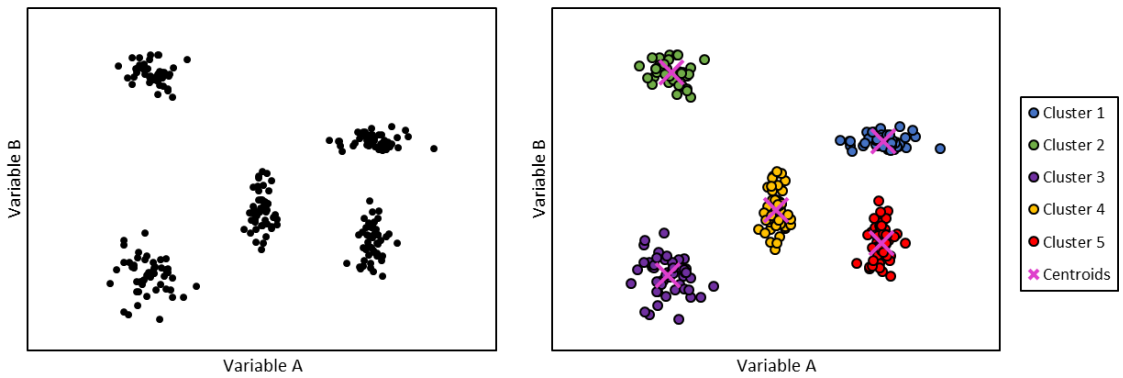

Figure 3.2: Illustration of k-means clustering. Observations containing two variables each are partitioned into five separate clusters, yielding five centroids that approximate the entire dataset.

Like standard PCA, the kernel PCA subspace does not account for significant autocorrelations between observations. However, the dynamic PCA modification was already well-established for standard PCA, and was swiftly adopted for kernel PCA by Choi and Lee (2004). The resulting dynamic kernel PCA model effectively addressed the static limitation of standard kernel PCA.

3.5 Auto-encoders

Auto-encoders (AEs) are subtypes of artificial neural networks that find effective representations of inputs and reconstruct them accurately. They can therefore be applied as reconstruction-based one-class classifiers (Tax, 2001). Neural networks are models that can fit a nonlinear function to any degree of precision (Cybenko, 1989; Kramer, 1991). AEs (like neural networks) consist of stacked layers of fully connected neurons, with layers finding nonlinear representations of their respective inputs. AEs feature bottleneck layers where they are forced to find an effective nonlinear subspace of modelled data.

Kramer (1991) introduced AEs to process monitoring by modelling normal condition data from a simulated batch reactor to detect faults. Dong and Mcavoy (1996) modelled normal condition data from the Tennessee Eastman process simulation, and found AE model performance superior to standard PCA. An AE that dynamically modelled normal condition data from the Tennessee Eastman process was used by Chen and Liao (2002) to remove nonlinearities from observations to make data more suitable for PCA. The Tennessee Eastman process was also used by Zhang et al. (2018) and Zhang et al. (2019) to demonstrate the nonlinear representation learning of AEs. Hu et al. (2020) used an AE to recognize specific faults in a simulated coal mill system.

3.5.1 AE computations

AE architecture refers to the number of layers and neurons per layer in the network. Figure 3.3 presents a typical AE network architecture. Note the three hidden layers between the input (yellow) and output (green) layers. The bottleneck layer (red) has few neurons and represents the nonlinear subspace of modelled data. If the number of bottleneck neurons matches the true subspace dimensionality of the modelled data, then the AE is able to reject all observations that do not belong in this subspace (Tax, 2001).

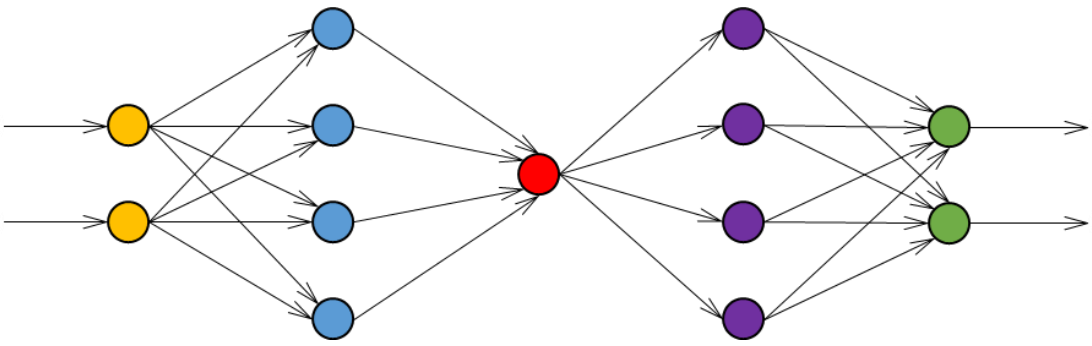

Figure 3.3: Typical AE architecture, with input (yellow), encoding (blue), bottleneck (red), decoding (purple) and output (green) layers

The simplest components of AEs are their neurons (Calli, 2017; Charte et al., 2020). Each neuron has a receptive field of multiple values, the neuron's output is a single value obtained by a linear combination of its receptive field before applying a nonlinear activation function. This is presented formally in equation 3-19:

$$\omega^{(J)} = \phi\left(\mathbf{W}^{(J)} \circ \boldsymbol{\alpha}^{(J)} + \beta^{(J)}\right) \qquad [3\text{-}19]$$

$\boldsymbol{\alpha}^{(J)}$ is a matrix representing the neuron's receptive field, adjusted linearly with a matrix of weights, $\mathbf{W}^{(J)}$, using a dot product operation (expressed by the $\circ$ operator) and adding a bias term, $\beta^{(J)}$. The nonlinear activation function, ϕ, yields the neuron output $\omega^{(J)}$. The outputs from multiple neurons serve as receptive fields to neurons in succeeding layers (Calli, 2017; Schmidhuber, 2015).

The gradient of ϕ plays a key role in AE optimization algorithms. Early nonlinear activation functions, like logistic sigmoid and hyperbolic tangent functions, struggled with numerical problems (most notable the vanishing gradient problem) that halted training in multilayer networks (Glorot et al., 2011). The ReLU activation function, presented in equation 3-20, is a function whose gradient does not vanish and has largely replaced logistic sigmoid and hyperbolic tangent activation functions in deep neural networks:

$$\phi_{ReLU}(x) = \max(x, 0) \qquad [3\text{-}20]$$

Backpropagation- and gradient descent algorithms train the AE (Ng, 2005). Backpropagation efficiently computes the contribution of individual weights to the loss function (Ng, 2017). Gradient descent algorithms update these weights according to their loss contributions. Equation 3-21 presents the stochastic gradient descent with momentum algorithm for optimizing a weight parameter, w:

$$w_{l+1} \coloneqq w_l - \eta\left(\frac{\partial\mathcal{E}\left(\mathbf{X}_t^{(s)}, \bar{\mathbf{X}}_t^{(s)}\right)}{\partial w_l}\right) + \gamma(w_l - w_{l-1}) \qquad [3\text{-}21]$$

w_l is the parameter value at the l^{th} training iteration, $\mathcal{E}\left(\mathbf{X}_t^{(s)}, \bar{\mathbf{X}}_t^{(s)}\right)$ is the loss function calculated over a randomly selected subset of the target data, $\mathbf{X}_t$. The third term introduces momentum (with momentum parameter γ) to the learning function, increasing the likelihood that the model will find globally optimized parameters. η is the learning rate, specifying how quickly w_l is updated according to its loss contribution.

The backpropagation- and gradient descent algorithms are crucial parts of AE optimization, and are discussed in greater detail in Appendix A – ANN Training. This appendix chapter also compares the different nonlinear activation functions used in ANNs to highlight the strength of ReLU in more detail and illustrate the vanishing gradient problem.

3.5.2 Limitations of fully connected AEs

The AEs introduced in this section all use fully connected layers; each neuron's receptive field is the entire output from the previous layer. Equation 3-19 shows that each neuron finds a single representative value of its receptive field. Complex, real-life patterns are rarely represented well with single values. This highlights the inherent flaw of using fully connected layers to learn from complex data: the output of a single neuron is simply unable to adequately represent a complex input (Wang et al., 2019). This

limitation manifests in fully connected AEs requiring many layers with many neurons to adequately learn faulty process conditions (Chen et al., 2020). Convolutional AEs, presented in the next section, are able to adequately represent complex patterns.

AEs require predefined network architectures, specifically optimized to the modelled data (Tax, 2001). They also require different design parameters to be specified beforehand, like choice of regularization function, choice of gradient descent algorithm and choice of loss function. Many of these design parameters also require parameters of their own. This introduces uncertainty to model evaluation as it cannot be determined if the model parameters are at a local or global optimum. This is in contrast to PCA models, whose model parameters are always at a global optimum (Ge et al., 2009).

3.6 Convolutional auto-encoders

Convolutional neural networks were developed for and completely outclass traditional feedforward networks in image recognition applications (Ko and Kim, 2020), but their adoption for process monitoring have been slowed by intrinsic differences between multivariate time series (MTS) and images. Convolutional neural networks developed for image recognition specifically preserve the image width and height dimensions, but MTS only have one such dimension: time (Ismail Fawaz et al., 2019). Still, preserving the temporal structure of MTS improves feature learning over traditional fully connected networks.

Ko and Kim (2020) showed that convolutional auto-encoders (CAEs) are suitable as reconstruction-based one-class classifiers to recognize process conditions in the Tennessee Eastman process, but their CAE processed MTS as though they were images. Wang et al. (2019) developed a novel one-dimensional convolutional auto-encoder for ECG signal compression. Chen et al. (2020) presented one-dimensional CAEs for fault recognition in both the Tennessee Eastman process and a fed-batch penicillin fermentation process, and demonstrated that one-dimensional CAEs are suitable for fault recognition.

3.6.1 CAE computations

A CAE is not computed fundamentally differently from fully connected AEs; both use neurons that characterize receptive fields with equation 3-19 and both typically use backpropagation- and gradient descent algorithms to optimize network parameters. In fact, fully connected AEs can be seen as a special case of CAE (Li et al., 2017). A simple CAE, consisting of two hidden layers with one neuron each, is presented in Figure 3.4.

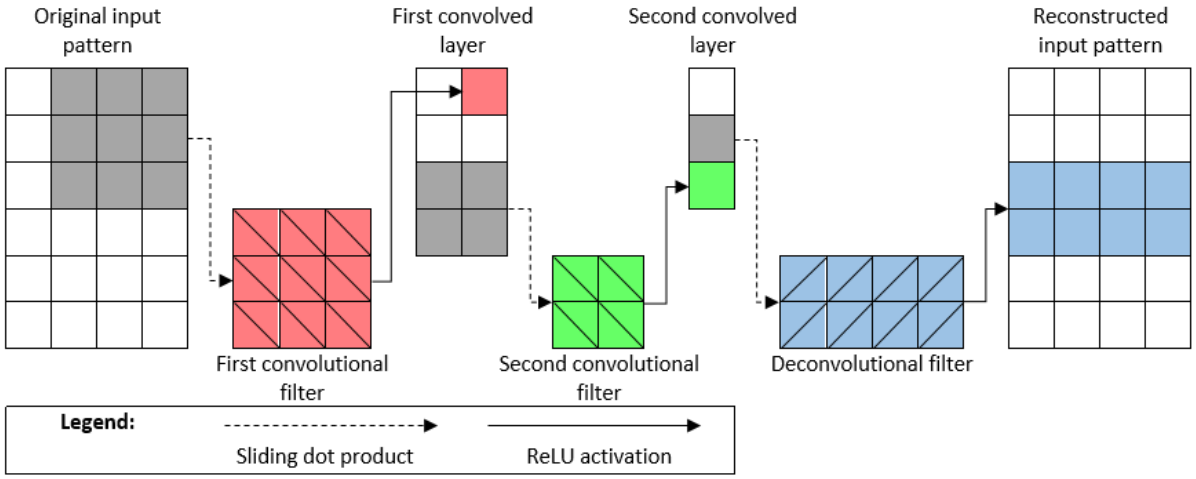

Figure 3.4: Simple 2-dimensional CAE with one convolutional filter per layer. Shaded areas represent the subsets of each layer output used as receptive field for subsequent convolutional filters.

CAEs use convolutional layers instead of fully connected layers. The receptive fields of neurons in convolutional layers are subsets of the outputs from preceding layers. These subsets (shaded in grey in Figure 3.4) are simpler than the entire layer output, and are easier to characterize by humble neurons. Each neuron calculates multiple outputs by sliding a filter over the preceding layer's outputs. These outputs are arranged in arrays to preserve the spatial structure of the preceding layer's output. Note the difference between filters in Figure 3.4: a convolutional filter down-samples its input (Calli, 2017), while a de-convolutional filter up-samples its input.

3.6.2 Limitations of convolutional AEs

CAEs inherit some of the limitations of conventional AEs; model parameters obtained through backpropagation- and gradient descent are not guaranteed to be the global optimum (Ge et al., 2009). Convolutional neural networks are typically outperformed by shallow, fully connected neural networks when few training observations of a specific class are available (Basha et al., 2020).

Another limitation of CAEs when applied to MTS stem from its novelty. Convolutional networks used for image processing typically extract simple, universal features (like edges) in the initial layers. Most images are composites of these simple features; this has spurred the development of pre-trained convolutional networks. Pre-trained networks are simple to tune for specific applications by optimizing the final layers for a specific task (Chen et al., 2020). Unfortunately, pre-trained convolutional networks trained to extract universal features from time series have not yet been published.

3.7 Pattern recognition performance

An FPR model is evaluated by how it flags observations. Table 3.1 defines four possible outcomes for an FPR model in a confusion matrix. A true positive indication is when the model correctly recognizes the presence of a fault, while a true negative indication is when the model correctly recognizes that the specific fault is not present (Salfner et al., 2010). Obviously, a good FPR model maximizes the number of true negative- and true positive indications, while minimizing the other outcomes.

Table 3.1: Confusion matrix for fault recognition.

	Fault present	No fault present	Total
Recognition	True positive – TP (recognition is correct)	False positive – FP (recognition is incorrect)	Total positives – I_1
No recognition	False negative – FN (lack of recognition is incorrect)	True negative – TN (lack of recognition is correct)	Total negatives – I_0
Total	Total faults – F_1	Total non-faults – F_0	Total observations – N

The outcomes summarized in Table 3.1 are converted to useful metrics that express FPR performance from different angles (Salfner et al., 2010). Accuracy (given in equation 3-22) is a straightforward metric that considers all possible prediction outcomes to quantify the likelihood of a correct outcome:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad [3\text{-}22]$$

Unfortunately, accuracy is a poor metric for unbalanced datasets (Jain et al., 2000; Tax, 2001). Other performance metrics, given in Table 3.2, are used in practice to express performance.

Table 3.2: Performance metrics for FPR models

Performance metric	Formula	Symbol
Precision	$\psi = \frac{TP}{TP+FP} = \frac{TP}{I_1}$	ψ
Sensitivity	$\varphi = \frac{TP}{TP+FN} = \frac{TP}{F_1}$	φ
Specificity	$\delta = \frac{TN}{TN+FP} = \frac{TN}{F_0}$	δ

Precision (ψ) expresses how likely it is that a recognition is correct, and is particularly important to plant operators as it directly relates to how many false alarms they have to deal with. Sensitivity (φ) and specificity (δ) indicate how likely it is that positive- and negative indications are correct (Jain et al., 2000).

The performance metrics presented so far require a defined recognition threshold. However, these recognition thresholds are determined empirically for reconstruction-based one-class classifiers, therefore section 3.7.1 discusses performance evaluation directly on discriminant values. Furthermore, recognitions based on single discriminant values exceeding a threshold are unreliable. Section 3.7.2 presents pattern recognition based on evaluating groups of discriminant values.

3.7.1 Receiver operating characteristic curve

The receiver operating characteristic (ROC) curve is a useful way of visualizing the performance of an FPR model based on the quality of discriminant values. An ROC curve plots sensitivity against specificity by moving the recognition threshold over discriminant values (Salfner et al., 2010). This is illustrated in Figure 3.5. The more the sensitivity curve approaches the top right corner of the chart, the closer the model is to a perfect classifier (with perfect sensitivity at complete specificity).

ROC-curve plots are useful for characterizing monitoring performance of individual models, but do not allow for clear comparisons between models (Tax, 2001); model comparisons based on ROC-curve shape would be very subjective. Furthermore, optimizing design parameters would be far simpler if a model's performance can be characterized in a single value. Luckily, the area under the ROC-curve (AUC_{ROC}) does just that by integrating sensitivity (φ) over specificity (δ):

$$AUC_{ROC} = \int_0^1 \varphi \, d\delta \qquad [3\text{-}23]$$

AUC_{ROC} reflects the probability that the FPR model would generate higher discriminant values for faulty observations than for fault-free observations (Salfner et al., 2010). It is therefore well suited for characterizing reconstruction-based FPR model performance.

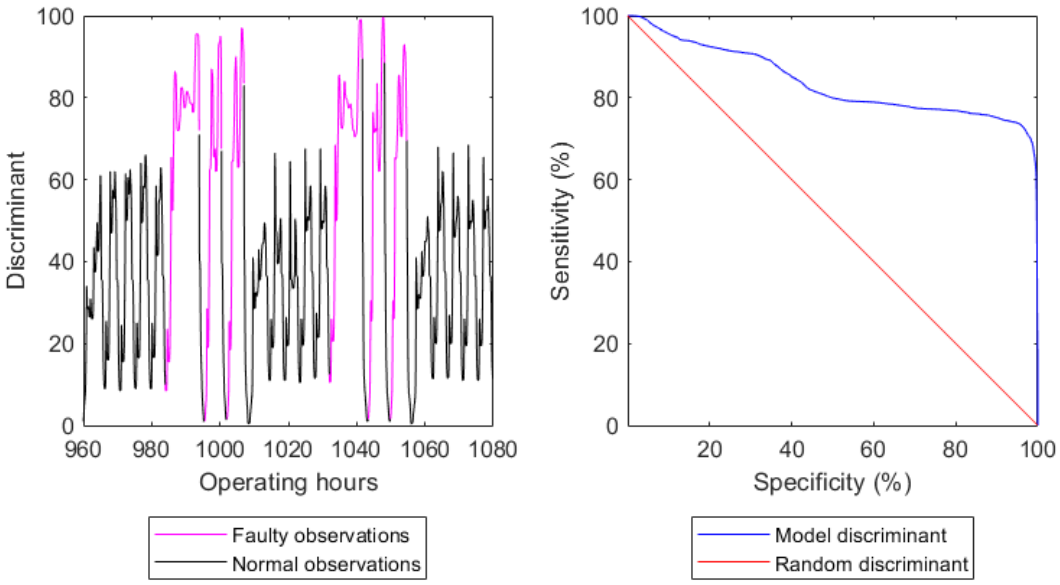

Figure 3.5: ROC-curve (right) generated from discriminant statistics (left) and random discriminant statistics. The ROC curve is obtained by calculating the sensitivity- and specificity at each discriminant value for the discriminant statistics in the left-hand figure.

Note the ROC-curve for a randomly generated discriminant to the right. It provides a useful baseline performance for evaluating an FPR model; the closer the model ROC is to a random discriminant and the poorer its performance. AUC_{ROC} also quantifies baseline FPR performance; $AUC_{ROC} = 0.5$ for a random discriminant.

Unfortunately, AUC_{ROC} can give misleading impressions of model performance; recognition performance at low specificity is rarely relevant but this region of the ROC curve inflates AUC_{ROC} (Dodd and Pepe, 2003). AUC_{ROC}-inflation is avoided by evaluating model performance only at relevant specificities:

$$AUC_{ROC}(\delta_1, \delta_2) = \int_{\delta_1}^{\delta_2} \varphi \, d\delta \qquad [3\text{-}24]$$

The relevant specificities in equation 3-24 (δ_1 and δ_2) depend on the application in question. The approach used for selecting δ_1 and δ_2 in this project is described in section 5.4.

3.7.2 Discriminant evaluation

Noisy discriminant values generated by FPR models make recognitions based on single values unreliable. Therefore effective FPR models evaluate windows of discriminant values. The standard approach is to recognize fault conditions if all values in the window exceed the recognition threshold (Sánchez-Fernández et al., 2018). This rejects false positives effectively, but a single false negative causes false negatives in the entire window.

The approach described by Singhal and Seborg (2002, 2000) considers that the number of observations that exceed the recognition threshold in a window (length l_w) follows a binomial distribution:

$$\alpha = \sum_{k=0}^{r} \binom{l_w}{k} \beta^k (1-\beta)^{l_w-k} \qquad [3\text{-}25]$$

β is the probability of an individual discriminant exceeding the threshold. α is the confidence that r values will exceed the recognition threshold. Figure 3.6 shows the recognition confidence, α, over the number of observations exceeding the recognition threshold, r, in a window with length l_w for $\beta = 0.5$.

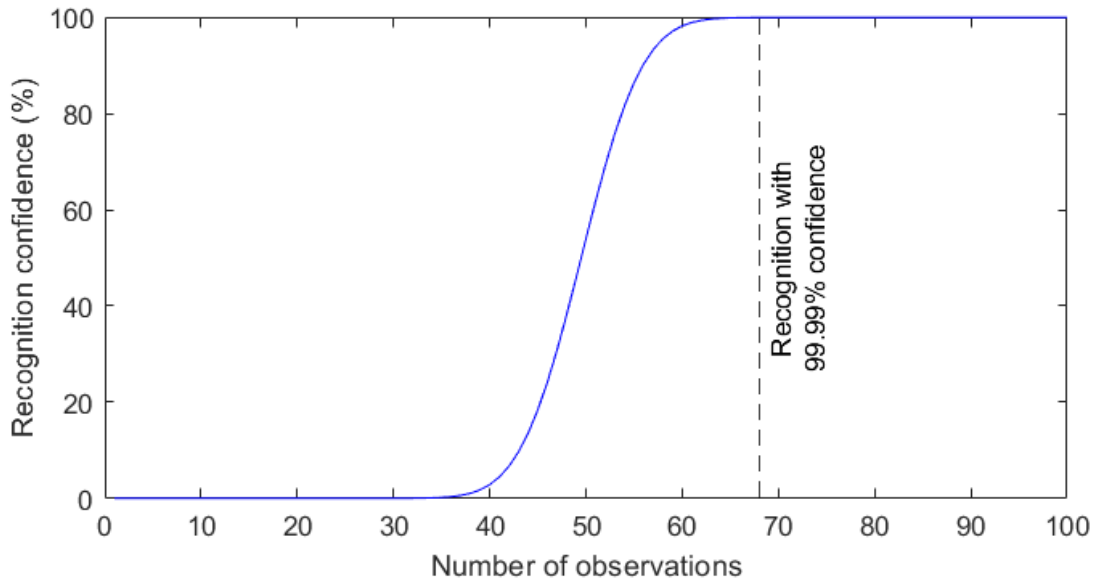

Figure 3.6: Recognition confidence for number of observations flagged in a recognition window.

Figure 3.6 shows that patterns can be recognized confidently from a window even if all observations do not exceed the recognition threshold. If β is assumed for discriminant values generated from faulty observations, then r_α out of l_w discriminant values should exceed the recognition threshold to recognize a fault pattern at the α-confidence level:

$$\begin{aligned} r_\alpha &= \max(r) \\ s.t.\ \ &\sum_{k=0}^{r} \binom{l_w}{k} \beta^k (1-\beta)^{l_w-k} < \alpha \end{aligned} \qquad [3\text{-}26]$$

4 FURNACE MODELLING APPROACH

This chapter presents the approach used to obtain a set of ODEs to simulate blowback data for FPR model development and evaluation, and addresses the first objective identified for this project. Section 4.1 discusses which distinct furnace zones are considered to represent the furnace interior. Section 4.2 discusses the assumptions made to facilitate modelling. Section 4.3 presents the model derivation using conventional mass- and energy balances over the zones identified in section 4.1. Section 4.4 describes the approach used to model gas flux through the concentrate bed in detail to inform the reader of how blowbacks are simulated in this model. Sections 4.5 and 4.6 presents- and evaluates the simulated data generated by this model for use in later model development. Section 4.7 briefly discusses how this model is implemented in MATLAB.

4.1 Model overview

This model approximates the submerged arc furnace interior with the distinct zones presented in Figure 4.1. Note that the reaction gas zone is disseminated over the bulk- and smelting concentrate zones, as it occupies the voids in the concentrate bed:

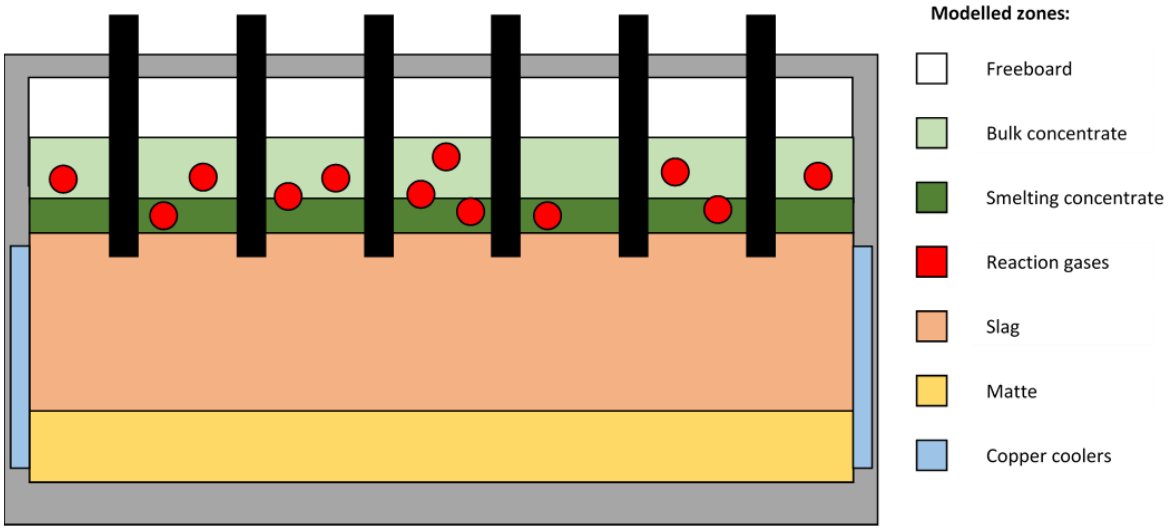

Figure 4.1: SAF model layout, with distinct bulk- (light green) and smelting concentrate (dark green), liquid slag- (beige) and matte (gold), trapped reaction gas (red), cooling units (blue) and freeboard (white) zones

4.1.1 Heat transfer considerations

The modes of heat transfer considered when deriving the model are presented below, with motivation where necessary. Heat transfer modes that featured most prominently in SAF models published in literature are favoured for inclusion in the derived model.

H1 Across the matte- and slag interface. This type of heat transfer was considered by all the literature SAF models reviewed for this model derivation.

H2 Between the top of the concentrate bed and the furnace freeboard. This heat transfer mechanism was considered by Pan et al. (2011) and played a key role in modelling dynamic freeboard behaviour in the EAF model by Logar et al. (2012a, 2012b).

H3 From the slag zone to the smelting concentrate zone. This form of heat transfer is crucial in the SAF energy balance, as most energy supplied to the furnace is consumed in smelting reactions (Sheng et al., 1998a, 1998b).

H4 Between trapped reaction gases and the concentrate bed. Reaction gases generated in the concentrate zone play a key role in transferring heat throughout the concentrate bed (Pan et al., 2011).

H5 Between cooling units and the matte- and slag zones. Heat removed from the furnace bath zones through side-wall coolers play a key role in furnace behaviour, and was included by Ritchie and Eksteen (2011).

H6 From the slag zone to newly-formed droplets descending from the smelting concentrate. This form of heat transfer plays a key role in the energy balance over the slag zone, and was modelled by Pan et al. (2011).

H7 From descending matte droplets to the matte zone. This heat transfer mechanism contributes greatly to the energy balance over the matte zone (Eksteen, 2011).

H8 From the bulk concentrate to newly charged concentrate. The reviewed SAF models omitted concentrate charging, but the EAF model by Logar et al. (2012a, 2012b) highlighted this heat transfer mechanism's importance in lowering the bulk concentrate temperature.

H9 From reaction gases escaping the concentrate bed to the furnace freeboard. Reaction gases from the concentrate are at a far higher temperature than the freeboard, therefore the heat they transfer to the freeboard influence freeboard behaviour, including pressure.

4.1.2 Mass transfer considerations

This model considers the following mass transfer mechanisms to approximate furnace behaviour. Again, motivation is provided where necessary.

M1 Matte- and slag droplets descending from the concentrate smelting zone.

M2 Concentrate charged to the furnace.

M3 Matte- and slag tapped from the furnace.

M4 Reaction gases released through desulphurization and electrode oxidation into concentrate.

M5 Reaction gases escaping from the concentrate to the furnace freeboard. Blowbacks occur when the freeboard gauge pressure becomes positive, therefore this form of mass transfer should naturally be included in a blowback-emulation model.

M6 Mixing of the bulk- and smelting concentrate zones. Pan et al. (2011) and Sheng et al. (1998a, 1998b) highlighted the importance of distinguishing between the bulk- and smelting concentrate zones when modelling SAF interiors, and this form of mass transfer approximates the interaction between these distinct zones.

M7 Freeboard gases transferred to/from the atmosphere.

4.2 Assumptions to facilitate modelling

The assumptions made to facilitate modelling are presented in this section, with motivation where necessary. These assumptions allow a set of ODEs to be derived to approximate an SAF.

4.2.1 Temperature profile

A1 Furnace zones have homogeneous temperatures. Reaction gases within the concentrate are at the same temperature as the smelting concentrate zone. The mass of reaction gases in the concentrate is small relative to the smelting concentrate, therefore its temperature can be lumped together with the smelting concentrate.

A2 Newly-formed droplets are at the same temperature as the smelting concentrate zone. Matte droplets entering the matte zone are at thermal equilibrium with the slag zone.

4.2.2 Zone composition

A3 The furnace interior contains lumped slag components (silicates- and oxides), lumped matte components (sulphides), lumped components that can undergo sulfurization (i.e. FeS_2), lumped reaction gases (SO_2 and CO_2) and lumped atmospheric air (O_2 and N_2). This greatly reduces the degrees of freedom by limiting the number of independent variables that have to be modelled.

A4 The bulk- and smelting concentrate zones contains slag, -matte and sulfurized components. Gases trapped in the bulk concentrate contains only reaction gases. The slag- and matte zones contain only slag- and matte components. The furnace freeboard contains reaction gases and air. Each zone has a homogeneous composition.

4.2.3 Furnace reactions

A5 Furnace reactions other than desulphurization, electrode oxidation and smelting are omitted. All desulphurization reactions are lumped together. Electrode oxidation reactions are lumped together. These reactions yield completely oxidized reaction gases. Desulphurization reactions also form matte components. Smelting reactions yield matte- and slag components; sulfurized matte components do not melt.

A6 Desulphurization reactions have irreversible, first order kinetics. Electrode oxidation reactions have zeroth order kinetics. Matte- and slag components melt at the same temperature. Smelting rate is determined by the ratio of smelting concentrate temperature to the concentrate melting temperature, and the heat transferred to the smelting concentrate from the slag zone. This mechanism was used by Bekker et al. (2000) and Logar et al. (2012a, 2012b) to dynamically simulating smelting rate in an ODE model.

A7 Chromite spinel formation is ignored. This is a valid assumption if the modelled slag temperature is high enough and electrode immersion is sufficient (Ritchie and Eksteen, 2011).

4.2.4 Heat generation- and transfer

A8 All heat generation occurs through Joule heating of the slag zone. This assumption was made for the SAF models by Bezuidenhout et al. (2009), Ritchie and Eksteen (2011) and Pan et al. (2011). Electrical energy is transferred to the slag zone by applying a constant voltage over the slag, as the CFD model by Bezuidenhout et al. (2009) found that omitting the time variation in energy supplied to the SAF did not impact model accuracy. The slag resistivity varies linearly with temperature.

A9 All heat transfer between zones occurs through convection. Heat transfer through the furnace refractory is omitted. Convection heat transfer is governed by constant heat transfer coefficients. Heat transferred from electrode oxidization gases to the smelting concentrate is lumped together with heat transferred from the slag through convection.

A10 Only the heat of smelting reactions are considered. All other reaction heats are omitted. This assumption is supported by the model developed by Sheng et al. (1998a, 1998b), who found that smelting reaction heats consume most of the energy supplied to the SAF.

4.2.5 Mass transfer

A11 All matte- and slag droplets report immediately to the matte- and slag zones, respectively. The retention time of matte in the slag zone was calculated through mechanistic modelling by Eksteen (2011) to be very low. Note that heat transferred to matte droplets is still considered by this model.

A12 First order volumetric mixing governs mass transfer from the bulk- to smelting concentrate.

A13 Reaction gas flux from the concentrate occurs in two distinct modes to facilitate blowbacks. In the first mode, flux is limited as if through a packed bed reactor. In the second, flux is limited by a constant.

4.3 Model derivation

The presented model is a set of 17 ordinary differential equations (ODEs). The state variables of these ODEs are presented in Table 4.1, along with the symbols used in the derived ODEs. A full list of symbols used in this derivation (including parameters) is given in Appendix D.

Table 4.1: State variables of ODEs

Bulk concentrate state variables:		$\boldsymbol{C(B)}$
1	Moles of slag component in bulk concentrate	$N_{C(B),XO}$
2	Moles of matte component in bulk concentrate	$N_{C(B),XS}$
3	Moles of sulfurized matte component in bulk concentrate	$N_{C(B),XS_2}$
4	Temperature of bulk concentrate	$T_{C(B)}$

Smelting concentrate state variables:		$\boldsymbol{C(S)}$
5	Moles of slag component in smelting concentrate	$N_{C(S),\,XO}$
6	Moles of matte component in smelting concentrate	$N_{C(S),\,XS}$
7	Moles of sulfurized matte component in smelting concentrate	$N_{C(S),\,XS_2}$
8	Temperature of smelting concentrate	$T_{C(S)}$
Reaction gases in concentrate state variables:		$\boldsymbol{C(R)}$
9	Moles of reaction gases trapped in concentrate	$N_{C(R)}$
Slag zone state variables:		$\boldsymbol{S}$
10	Moles of slag in the slag zone	N_S
11	Temperature of the slag zone	T_S
Matte zone state variables:		$\boldsymbol{M}$
12	Moles of matte in the matte zone	N_M
13	Temperature of matte zone	T_M
Furnace freeboard state variables:		$\boldsymbol{G}$
14	Moles of reaction gases in the furnace freeboard	$N_{G,R}$
15	Moles of air in the furnace freeboard	$N_{G,\,A}$
16	Temperature of the furnace freeboard	T_G
Cooling units state variables:		$\boldsymbol{W}$
17	Cooling water temperature	T_W

Equations 4-1 to 4-17 in sections 4.3.1 to 4.3.7 present the ODEs for state variables according to the furnace zone they model. Energy balances are used to derive the ODEs for temperature state variables, while component balances are used to derive ODEs for other state variables. Sections 4.3.8 and 4.3.9 elaborate on prominent expressions contained in the ODEs.

The expressions given here do not amount to a complete degrees of freedom (DOF) analysis of the ODE model; this is given in Appendix C. These expressions are meant to guide the reader to understanding how the ODEs represent prominent SAF behaviour. Note that reaction gas flux from the concentrate bed to the furnace freeboard (J_R) is a cornerstone of the proposed blowback mechanism and is discussed separately in section 4.4.

4.3.1 Bulk concentrate derivation

The component balance over the bulk concentrate, given in equations 4-1 to 4-3, is determined by the rate of concentrate charged to the furnace, $F_{charge}\ \left[\frac{mol}{s}\right]$, the rate of mixing with the smelting concentrate, $F_{mix}\ \left[\frac{mol}{s}\right]$, and the rate of desulphurization ($r_{F,\,C(B)}V_{C(B)}\ \left[\frac{mol}{s}\right]$).

$$\frac{dN_{C(B),\,XO}}{dt} = F_{charge,\,XO} - F_{mix,\,XO} \qquad [\,4\text{-}1\,]$$

$$\frac{dN_{C(B),\,XS}}{dt} = F_{charge,\,XS} - F_{mix,\,XS} + r_{F,\,C(B)}V_{C(B)} \qquad [\,4\text{-}2\,]$$

$$\frac{dN_{C(B),\,XS_2}}{dt} = F_{charge,\,XS_2} - F_{mix,\,XS_2} - r_{F,\,C(B)}V_{C(B)} \qquad [\,4\text{-}3\,]$$

Note that desulphurization reactions in the bulk concentrate convert sulfurized matte (XS_2) to matte (XS); this is why the desulphurization term appears in both balances with reversed signs. The rate of change in bulk concentrate temperature is determined by the total heat transferred to it, the energy required to heat charged concentrate to the bulk concentrate temperature and the heat required to increase the bulk concentrate temperature, $c_{P,\,C}N_{C(B)}\ \left[\frac{kJ}{K}\right]$:

$$\frac{dT_{C(B)}}{dt} = \frac{Q_{C(S):C(B)} + Q_{G:C(B)} + c_{P,\,C}\left(T_{charge} - T_{C(B)}\right)\cdot F_{charge}}{N_{C(B)}\cdot c_{P,\,C}} \qquad [\,4\text{-}4\,]$$

The first term in the numerator account for heat exchanged with the smelting concentrate and reaction gases in the concentrate. The second term accounts for heat exchanged with the furnace freeboard. The final term in the numerator accounts for heat used to increase the charged concentrate temperature to the bulk concentrate temperature.

4.3.2 Smelting concentrate derivation

Equations 4-5 to 4-7 presents the components balance over the smelting concentrate zone. This component balance depends on the rate of mixing with the bulk concentrate, the rate of desulphurization in the smelting concentrate ($r_{F,\,C(S)}V_{C(S)}\ \left[\frac{mol}{s}\right]$) and concentrate melting rate ($F_{melt}\ \left[\frac{mol}{s}\right]$):

$$\frac{dN_{C(S),\,XO}}{dt} = F_{mix,\,XO} - F_{melt,\,XO} \qquad [\,4\text{-}5\,]$$

$$\frac{dN_{C(S),\,XS}}{dt} = F_{mix,\,XS} - F_{melt,\,XS} + r_{F,\,C(S)}V_{C(S)} \qquad [\,4\text{-}6\,]$$

$$\frac{dN_{C(S),\,XS_2}}{dt} = F_{mix,\,XS_2} - r_{F,\,C(S)}V_{C(S)} \qquad [\,4\text{-}7\,]$$

Equation 4-8 presents the rate of change in smelting concentrate temperature:

$$\frac{dT_{C(S)}}{dt} = \frac{Q_{S:C(S)}\left(1 - \frac{T_{C(S)}}{T_{C,\,melt}}\right) - Q_{C(S):C(B)} + \left(T_{C(B)} - T_{C(S)}\right)\left(F_{mix}c_{P,\,C} + r_{F,\,C(B)}V_{C(B)}c_{P,\,G}\right)}{N_{C(S)}c_{P,\,C}} \qquad [\,4\text{-}8\,]$$

The first term in the numerator, $Q_{S:C(S)}\left(1 - \frac{T_{C(S)}}{T_{C,\,melt}}\right)\ [kW]$ is the heat transferred from the slag used to heat the melting concentrate. This term decreases as the smelting zone temperature approaches the

melting temperature. The second term represents the heat transferred to the bulk concentrate. The final term expresses heating of mixed concentrate and desulphurization gas phase products from the bulk concentrate.

4.3.3 Reaction gases in the concentrate derivation

Gases within the concentrate zone contain only one component, and its temperature is lumped with the smelting concentrate zone. Therefore only a single ODE is needed to represent this zone:

$$\frac{dN_{C(R)}}{dt} = r_{F,\,C(B)}V_{C(B)} + r_{F,\,C(S)}V_{C(S)} + r_C - J_R A \qquad [4\text{-}9]$$

$r_{F,\,C(B)}V_{C(B)}$ and $r_{F,\,C(S)}V_{C(S)}$ represents gases generated by desulphurization reactions in the bulk- and smelting concentrate zones, respectively. $r_C\ \left[\frac{mol}{s}\right]$ is the rate of electrode oxidation. $J_R A\ \left[\frac{mol}{s}\right]$ is the rate at which reaction gases escape to the furnace freeboard, and forms a key part of how this model emulates blowbacks (discussed in section 4.4).

4.3.4 Slag zone derivation

The slag zone only contains slag components, therefore two ODEs are sufficient to characterize this zone:

$$\frac{dN_S}{dt} = F_{melt,\,XO} - F_{tap,\,S} \qquad [4\text{-}10]$$

$F_{tap,\,S}\ \left[\frac{mol}{s}\right]$ is the rate at which slag is tapped from the furnace.

$$\frac{dT_S}{dt} = \frac{Q_J + Q_{M:S} + Q_{W:S} - Q_{S:C(S)} + (T_{C,\,melt} - T_S)(F_{melt,\,XO}c_{P,\,S} + F_{melt,\,XS}c_{P,\,M})}{N_S c_{P,\,S}} \qquad [4\text{-}11]$$

The first term in the numerator, $Q_J\ [kW]$, is the heat generated through Joule heating in the slag zone. The subsequent three terms represent heat transferred with the matte, cooling units- and smelting concentrate, respectively. The final term represents the heat used to heat matte- and slag droplets to the slag zone temperature.

4.3.5 Matte zone derivation

The matte zone contains a single component and can be characterized in two ODEs:

$$\frac{dN_M}{dt} = F_{melt,\,XS} - F_{tap,\,M} \qquad [4\text{-}12]$$

$F_{tap,\,M}\ \left[\frac{mol}{s}\right]$ is the rate at which matte is tapped from the furnace.

$$\frac{dT_M}{dt} = \frac{Q_{W:M} - Q_{M:S} + (T_S - T_M)F_{melt,\,XS}c_{P,\,M}}{N_M c_{P,\,M}} \qquad [4\text{-}13]$$

The first two terms in the numerator represents heat exchanged with cooling units and the slag zone, respectively. This final numerator term represents the heat exchanged with incoming matte droplets as they mix with the matte zone.

4.3.6 *Furnace freeboard derivation*

The furnace freeboard contains two components: air and reaction gases. The resulting ODEs for the component balance over the freeboard are given below:

$$\frac{dN_{G,A}}{dt} = F_{neg.P} - F_{pos.P,A} - F_{ext,A} \quad [4\text{-}14]$$

$$\frac{dN_{G,R}}{dt} = J_R A - F_{pos.P,R} - F_{ext,R} \quad [4\text{-}15]$$

$F_{neg.P}\ \left[\frac{mol}{s}\right]$ represents the air drawn in from the atmosphere when the freeboard pressure is negative, therefore it is not replicated for reaction gases in the freeboard. $F_{pos.P}\ \left[\frac{mol}{s}\right]$ represents gas blown out of the furnace when pressure is positive, while $F_{ext}\ \left[\frac{mol}{s}\right]$ represents gas extracted from the furnace to avoid blowbacks.

$$\frac{dT_G}{dt} = \frac{c_{P,G}\left[\left(T_{C(S)} - T_G\right)J_R A + \left(T_{atm} - T_G\right)F_{neg.P}\right] - Q_{G:C(B)}}{N_G c_{P,G}} \quad [4\text{-}16]$$

The furnace freeboard temperature is affected by heat convection with the bulk concentrate, $Q_{G:C(B)}$, and heat transferred to reaction gases escaping the concentrate and air drawn into the freeboard. These last two factors are represented by the first term in the numerator of equation 4-16.

4.3.7 *Cooling units derivation*

A constant flow of cooling water through the cooling units is assumed. Therefore, only an energy balance is needed to characterize the cooling units-zone:

$$\frac{dT_W}{dt} = \frac{c_{P,W}\left(T_{W,0} - T_W\right)F_W - Q_{W:M} - Q_{W:S}}{N_W c_{P,W}} \quad [4\text{-}17]$$

The first term in the numerator represents the heat transferred to cooling water entering the cooling units. The last two terms represent heat exchanged with the matte- and slag zones, respectively.

4.3.8 *Mass transfer- and reaction rate expressions*

This section elaborates on prominent mass transfer and reaction rate expressions presented in the ODEs. The first of these is the rate of concentrate components charged to the furnace:

$$F_{charge,i} = F_{charge} x_{charge,i} \quad [4\text{-}18]$$

$x_{charge,i}$ is the mole fraction of component i in the charged concentrate, and is an independent variable. F_{charge} is varied in the presented model to keep the concentrate bed height between operating thresholds:

$$\begin{gathered} if\ F_{charge} = 0\ and\ L_C < L_{C,\,lower\ lim} \\ F_{charge} \leftarrow F_{charge,0} \\ else\ if\ F_{charge} = F_{charge,0}\ and\ L_C \geq L_{C,\,upper\ lim} \\ F_{charge} \leftarrow 0 \end{gathered} \quad [4\text{-}19]$$

$L_{C,\,upper\,lim.}\,[m]$ and $L_{C,\,lower\,lim.}[m]$ are the upper- and lower limits of the concentrate bed thickness. $F_{charge,\,0}\,\left[\frac{mol}{s}\right]$ is the concentrate charging rate used to increase the concentrate bed thickness. Volumetric mixing governs component transfer from the bulk- to smelting concentrate:

$$F_{mix,\,i} = k_v V_{C(B)} C_{C(B),\,i} \qquad [\,4\text{-}20\,]$$

$k_v\,\left[\frac{1}{s}\right]$ is the volumetric mixing constant, $V_{C(B)}\,[m^3]$ is the volume of bulk concentrate and $C_{C(B),\,i}\,\left[\frac{mol}{m^3}\right]$ is the concentration of component i in the bulk concentrate. Desulphurization reactions are considered separately for the bulk- and smelting concentrate zones:

$$r_{F,\,j} = k_F e^{\left(-\frac{E_{A,\,F}}{RT_j}\right)} C_{j,\,XS_2} \qquad [\,4\text{-}21\,]$$

$k_F\,\left[\frac{1}{s}\right]$ is the rate constant of the desulphurization reaction, $E_{A,\,F}\,\left[\frac{J}{mol\cdot K}\right]$ is the activation energy of the desulphurization reaction, $R\,\left[\frac{J}{mol\cdot K}\right]$ is the universal gas constant. $T_j\,[K]$ is the temperature of the concentrate zone in question. $C_{j,\,XS_2}\,\left[\frac{mol}{m^3}\right]$ is the sulfurized matte concentration of the concentrate zone in question. Electrode oxidation is described by the following zeroth order reaction rate:

$$r_C = k_C e^{\left(-\frac{E_{A,\,C}}{RT_S}\right)} \qquad [\,4\text{-}22\,]$$

$k_C\,\left[\frac{1}{s}\right]$ is the electrode oxidation rate constant, $E_{A,\,C}\,\left[\frac{J}{mol\cdot K}\right]$ is the electrode oxidation reaction activation energy and $T_S\,[K]$ is the temperature of the slag zone.

A component's smelting rate, $F_{melt,\,i}\,\left[\frac{mol}{s}\right]$ is a function of the temperature of the smelting concentrate zone, $T_{C(S)}\,[K]$, the heat transferred to the smelting concentrate from the slag zone, $Q_{S:C(S)}\,[kW]$, and the composition of the smelting concentrate zone.

$$F_{melt,\,i} = \frac{Q_{S:C(S)}\left(\frac{T_{C(S)}}{T_{C,\,melt}}\right)}{\lambda_C + c_{P,\,C}\left(T_{C,\,melt} - T_{C(S)}\right)} \cdot \frac{N_{C(S),\,i}}{N_{C(S),\,XO} + N_{C(S),\,XS}} \qquad [\,4\text{-}23\,]$$

$\lambda_C\,\left[\frac{kJ}{mol}\right]$ is the latent heat of fusion of the concentrate, $c_{P,\,C}\,\left[\frac{kJ}{mol\cdot K}\right]$ is the heat capacity of the concentrate and $T_{C,\,melt}\,[K]$ is the concentrate melting temperature. Note that sulfurized components are not included when calculating smelting product composition.

Equation 4-24 quantifies the molar flow of atmospheric air into the furnace if the freeboard gauge pressure is negative, and 4-25 quantifies the molar flow of freeboard gases to the atmosphere if the freeboard gauge pressure is positive:

$$F_{neg.\,P} = \begin{cases} k_{PR}(P_{atm} - P_G)\ if\ P_{atm} \geq P_G \\ 0\ if\ P_{atm} < P_G \end{cases} \qquad [\,4\text{-}24\,]$$

$$F_{pos.\,P,\,i} = \begin{cases} \frac{k_{PR} N_{G,\,i}(P_G - P_{atm})}{N_{G,\,A} + N_{G,\,R}}\ if\ P_G \geq P_{atm} \\ 0\ if\ P_G < P_{atm} \end{cases} \qquad [\,4\text{-}25\,]$$

k_{PR} $\left[\frac{mol}{Pa \cdot s}\right]$ is the ratio of molar flow to pressure difference between the furnace freeboard and surroundings. P_G $[Pa]$ is the pressure of the freeboard (obtained with the ideal gas law), and P_{atm} $[Pa]$ is the pressure of the surrounding atmosphere. Note that equation 4-25 computes the composition of the freeboard; this is because the freeboard contains reaction gases while the atmosphere hopefully does not. Finally, equation 4-26 quantifies the molar flow due to gas extraction from the furnace:

$$F_{ext,\,i} = \frac{k_{PE} N_{G,\,i} (P_G - P_{ext})}{N_{G,\,A} + N_{G,\,R}} \qquad [\text{4-26}]$$

k_{PE} $\left[\frac{mol}{Pa \cdot s}\right]$ is the ratio of molar flow to the pressure difference between the furnace freeboard and the forced draught, P_{ext} $[Pa]$.

4.3.9 Heat generation- and transfer expressions

Equation 4-27 presents the Joule heating expression used to quantify heat generation in the slag zone:

$$Q_J = \frac{V_{electrodes}^2}{R_0 (1 + \alpha [T_S - T_{S,\,0}])} \qquad [\text{4-27}]$$

$V_{electrode}$ $[V]$ is the voltage applied over the slag zone. R_0 $\left[\frac{V^2}{kW}\right]$ is the slag resistivity when the slag temperature is $T_{S,\,0}$ $[K]$. α $\left[\frac{1}{K}\right]$ is the temperature coefficient of resistance, describing how resistivity varies linearly with temperature. Heat transfer between zones occurs through convection, quantified for zones i and j by equation 4-28:

$$Q_{i:j} = h_{i:j} A_{i:j} (T_i - T_j) \qquad [\text{4-28}]$$

$h_{i:j}$ $\left[\frac{kW}{m^2 \cdot K}\right]$ is the convection heat transfer coefficient and $A_{i:j}$ $[m^2]$ the contact area between zones i and j. $h_{i:j}$ is constant for all zones, but $A_{i:j}$ can vary between matte- and slag zones and the cooling units as the level of liquid matte- and slag varies. Equation 4-29 quantifies convection heat transfer between zones with varying contact areas:

$$Q_{W:S/M} = h_{W:S/M} p_{SAF} L_{S/M} (T_W - T_{S/M}) \qquad [\text{4-29}]$$

p_{SAF} $[m]$ is the perimeter of the furnace bath, and $L_{S/M}$ is the height of the liquid matte- or slag.

4.4 Blowback mechanism

Literature does not provide established mechanisms for simulating blowbacks. The steelmaking EAF models by Bekker et al. (2000) and Logar et al. (2012a, 2012b) provide useful ways of modelling mass transfer between the freeboard and atmosphere during a blowback, but do not give suggestions on how blowbacks are caused. This section proposes a mechanism to emulate blowbacks, and illustrates how this mechanism is implemented in ODE expressions.

Reaction gases accumulate in concentrate bed voids (described by equation 4-9). This causes pressure build-up in the concentrate bed voids, calculated with equation 4-30:

$$P_{C(R)} = \frac{N_{C(R)} R T_{C(S)}}{\varepsilon_C V_C} \qquad [\text{4-30}]$$

$N_{C(R)}\ [mol]$ is the amount of reaction gases accumulated in the concentrate voids. The total volume of these voids is expressed by the denominator term ($\varepsilon_C V_C\ [m^3]$). The ideal gas law used by equation 4-30 is justified by the high temperatures in the concentrate, and the pressures not reaching a sufficient level for intermolecular forces to become significant. A pressure gradient forms over the concentrate bed, causing gas to flux to the freeboard zone as though through a packed bed reactor:

$$J_{R,\,PBR} = \frac{k_{PBR}}{L_C}\Delta P_{C(R):G} \qquad [\text{4-31}]$$

$k_{PBR}\ \left[\frac{mol}{m \cdot Pa \cdot s}\right]$ is the flux coefficient, $L_C\ [m]$ is the height of the concentrate bed and $\Delta P_{C(R):G}\ [Pa]$ the pressure difference between trapped reaction gases and the furnace freeboard.

Equation 4-31 shows that thicker concentrate beds (higher L_C) result in high $\Delta P_{C(R):G}$ for the same reaction gas flux, $J_{R,\,PBR}$. The concentrate bed ruptures if $\Delta P_{C(R):G}$ becomes too high. This critical pressure difference, $\Delta P_{crit}\ [Pa]$, is a function of the concentrate bed height:

$$\Delta P_{crit} = \rho_{C,\,bulk} g L_C \qquad [\text{4-32}]$$

$\rho_{C,\,bulk}\ \left[\frac{kg}{m^3}\right]$ is the concentrate bulk density, $g\ \left[\frac{m}{s^2}\right]$ is the gravitational constant, and $L_C\ [m]$ is the total concentrate bed height. A ruptured concentrate bed does not constrain reaction gas flux as through a packed bed reactor. The ruptures cause channels to form, and reaction gases bypass the bed to escape to the freeboard:

$$J_{R,\,Ch} = k_{Ch}\Delta P_{C(R):G} \qquad [\text{4-33}]$$

$k_{Ch}\left[\frac{mol}{Pa \cdot s}\right]$ is the channeling flux coefficient. $J_{R,\,Ch}$ is not constrained by concentrate bed thickness, and will therefore be larger than $J_{R,\,PBR}$. The flux term used in the model ODEs, J_R, has to be able to switch between these modes of flux as bed ruptures form- and dissipate. Equation 4-34 shows how $\Delta P_{C(R):G}$ and ΔP_{crit} are used to switch between different modes of flux.

$$\begin{gathered} if\ J_R = J_{R,\,PBR}\ and\ \Delta P_{C(R):G} > c\Delta P_{crit} \\ J_R \leftarrow J_{R,\,Ch} \\ if\ J_R = J_{R,\,Ch}\ and\ \Delta P_{C(R):G} \leq \Delta P_{crit} \\ J_R \leftarrow J_{R,\,PBR} \end{gathered} \qquad [\text{4-34}]$$

c is a hysteresis term that prevents the flux term from switching back-and-forth between $J_{R,\,PBR}$ and $J_{R,\,Ch}$ when the pressure difference becomes critical.

Using equation 4-34, flux will behave as through a packed bed reactor ($J_{R,\,PBR}$) until the bed rupturing pressure ($c\Delta P_{crit}$) is reached. At this point, flux will behave as though there are wide channels in the concentrate bed for reaction gases to pass through ($J_{R,\,Ch}$). This causes lots of reaction gases to release into the freeboard, raising its pressure and causing a blowback. These channels exist until the pressure difference becomes insufficient ($\Delta P_{C(R):G} \leq \Delta P_{crit}$). At this point, bed channels close and flux is once again constrained as though through a packed bed reactor.

4.5 Developed furnace model validation

The simulated data generated by the furnace model is validated in this section. The model is validated by comparing key outputs to production details reported in literature for Anglo Platinum's Polokwane smelter by Crundwell et al. (2011a) and for Lonmin's Furnace No. 1 by Eksteen et al. (2011). The validation presented in this section is intended to justify using the output data generated by the developed furnace model for FPR model evaluation. Signals generated by the developed model will be validated based on how well they align, compared to other PGM furnace models, with the production details reported for industrial furnaces in Table 4.2:

Table 4.2: Production details for Anglo Platinum's Polokwane smelter and Lonmin's Furnace No. 1

Validation variable	Value	Industry furnace	Source
Slag zone temperature	$1873\ K$	Anglo Platinum's Polokwane smelter	Crundwell et al. (2011a)
Matte zone temperature	$1823\ K$		
Power rating	$68\ MW$		
Throughput capacity	$87\ tonnes/h$		
Electrode oxidation rate	$3\ kg/ton\ throughput$		
Freeboard temperature	773 to $873\ K$	Lonmin's Furnace No. 1	Eksteen et al. (2011)

The slag temperature generated by the model over 7 days of simulated operation is presented in Figure 4.2, along with the average slag temperature reported for the Polokwane smelter by Crundwell et al. (2011a). Figure 4.2 also includes the average slag temperature reported for the PGM-smelting furnace models developed by Bezuidenhout et al. (2009), Eksteen (2011) and Ritchie and Eksteen (2011). Figure 4.3 presents the corresponding matte temperatures.

Note that the goal of this study is not an accurate furnace model, but to test FPR methods on a simulation with dynamic behaviour. Therefore this section only validates the furnace model using static values reported in literature. This is to show that the developed model aligns with the established models of submerged arc furnaces, and that the dynamics observed does not deviate so strongly as to render the signals generated implausible.

Figure 4.2 shows that the average slag temperature generated by the developed model is slightly lower than the average slag temperature found on the Polokwane smelter. However, when compared to the PGM-smelting furnace models by Bezuidenhout et al. (2009), Eksteen (2011) and Ritchie and Eksteen (2011), the developed model's slag temperature profile aligns closely with the slag temperature reported by Crundwell et al. (2011a) for Anglo Platinum's Polokwane smelter.

Figure 4.3 shows that while the average matte temperature generated by the developed model exceeds the average matte temperature reported by Crundwell et al. (2011a) for Anglo Platinum's Polokwane smelter, it falls within the spectrum of average matte temperatures modelled by Bezuidenhout et al. (2009), Eksteen (2011) and Ritchie and Eksteen (2011).

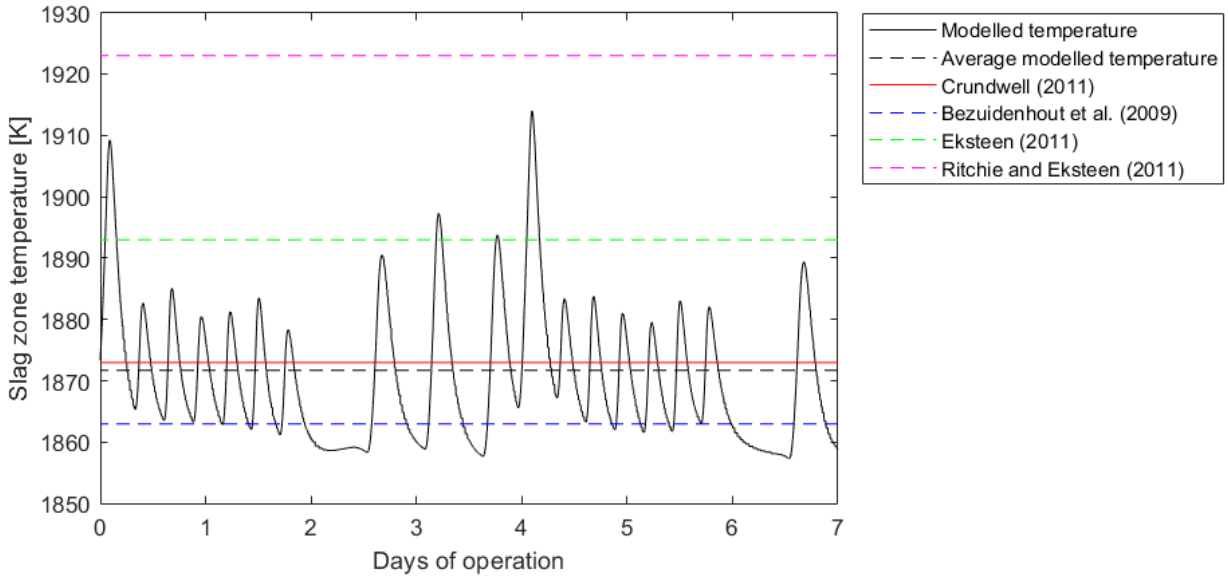

Figure 4.2: Slag temperature- and average slag temperature generated by the developed model over one week's simulation, with the average slag temperatures modelled by Bezuidenhout et al. (2009), Eksteen (2011) and Ritchie and Eksteen (2011) as well as the average slag temperature found on the Polokwane smelter.

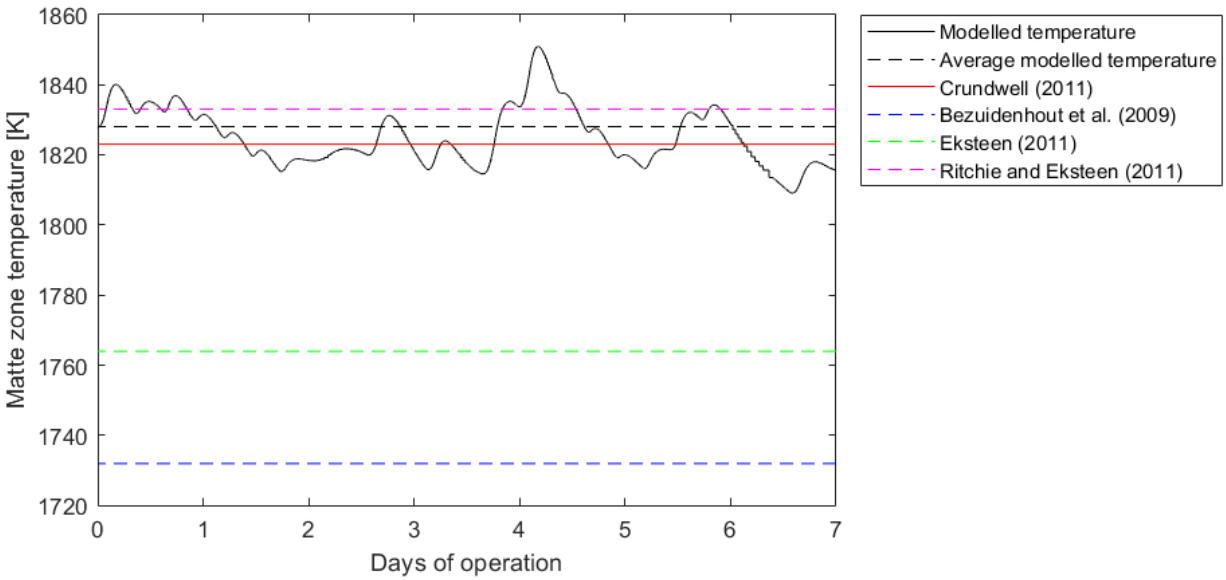

Figure 4.3: Matte temperature- and average matte temperature generated by the developed model over one week's simulation, with the average matte temperatures modelled by Bezuidenhout et al. (2009), Eksteen (2011) and Ritchie and Eksteen (2011) as well as the average matte temperature found on the Polokwane smelter.

Figure 4.4 presents the simulated power supply to the furnace model over seven days calculated with equation 4-27), as well as the average modelled power supply and the power rating of Anglo Platinum's Polokwane smelter.

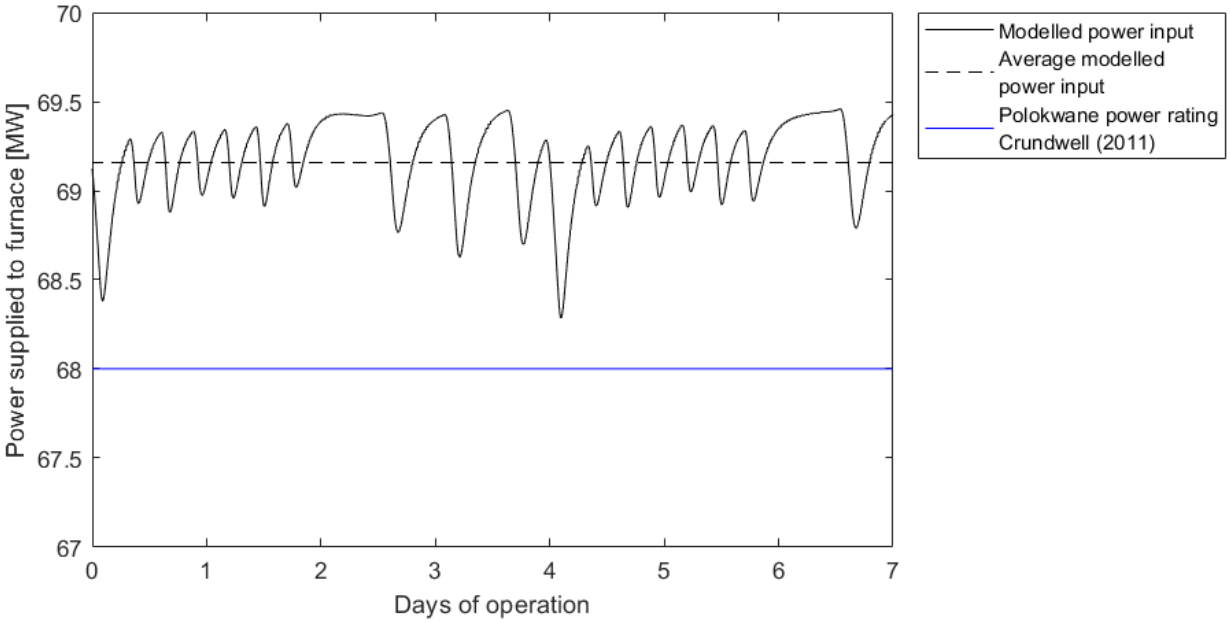

Figure 4.4: Power supplied to the modelled furnace over one week of simulated operation (solid black line), with the power rating for Anglo Platinum's Polokwane smelter reported by Crundwell et al. (2011a). The average power supply to the model is shown by the dashed black line.

Figure 4.4 shows that the modelled furnace's power use is higher than the power rating for the Polokwane smelter. Unfortunately, the power use for other PGM smelter models have not been reported in literature, therefore this discrepancy cannot be viewed in the context of other PGM smelter models. However, the average power use of the modelled furnace (69.2 MW) exceeds the power rating reported by Crundwell et al. (2011a) by less than 2%, therefore this discrepancy is acceptable. Figure 4.5 presents the modelled furnace throughput over seven days of simulated operation, as well as the capacity of Anglo Platinum's Polokwane smelter.

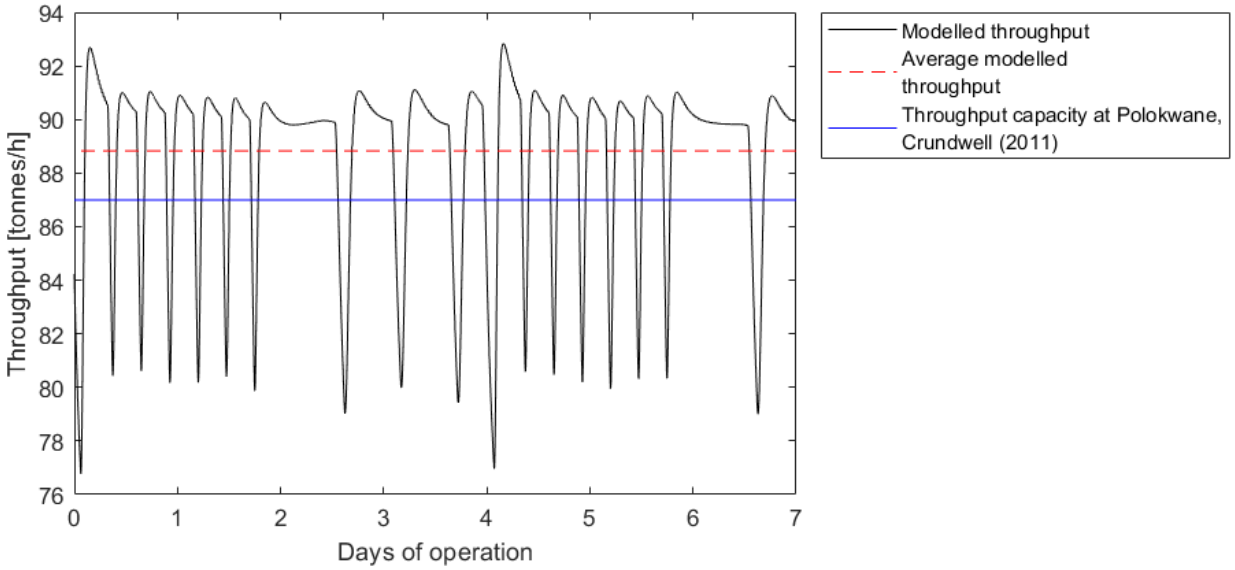

Figure 4.5: Modelled furnace throughput over seven days of simulated operation, compared with the capacity of the Polokwane smelter. The average modelled throughput is indicated by the red dashed line.

The average modelled throughput (88.8 tonnes per hour) of the furnace model exceeds the capacity of the Polokwane smelter (87 tonnes per hour) by approximately 2.1%. The throughput values for the reviewed PGM furnace models by Bezuidenhout et al. (2009), Eksteen (2011) and Ritchie and Eksteen (2011) have not been reported in literature, therefore the observed discrepancy cannot be contextualized with other furnace models. However, this discrepancy is considered small enough to be acceptable for the purposes of this project.

Figure 4.6 presents the electrode consumption rate per tonne of concentrate smelted over a week of simulated operation for the developed model, as well as the model's average electrode consumption rate. Figure 4.6 also presents the average electrode consumption rates reported by Crundwell et al. (2011a) for the Polokwane, Lonmin and Zimplats smelters.

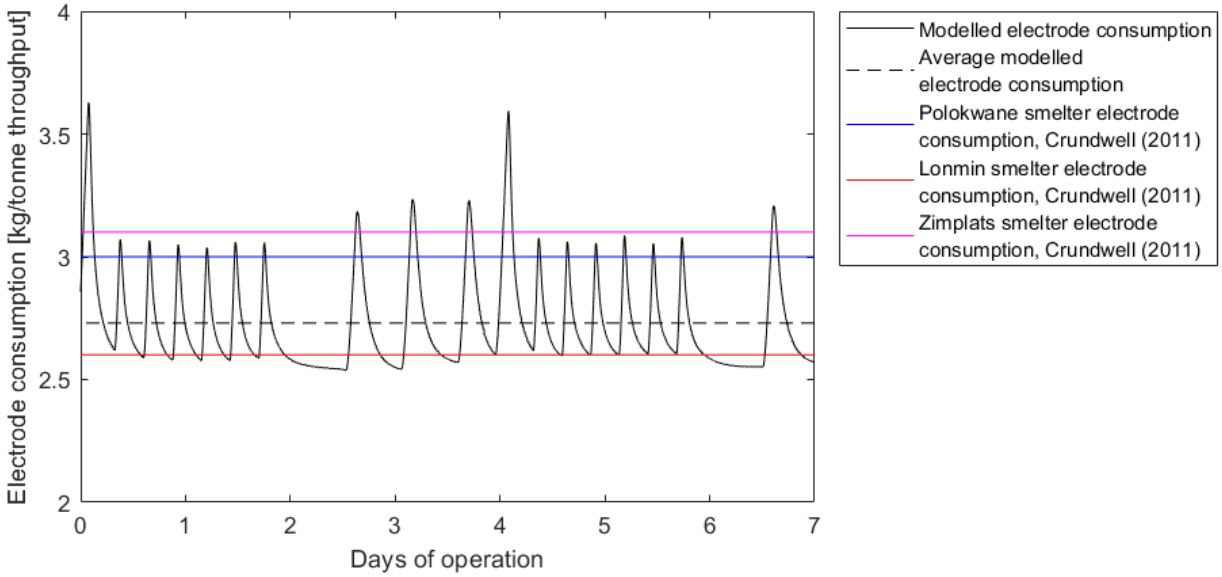

Figure 4.6: Modelled electrode consumption rate over a week of simulated operation, with the average electrode consumption rates for other furnaces reported by Crundwell et al. (2011a).

Figure 4.6 shows that the modelled electrode consumption rate (governed by equation 4-22) is generally lower than the average electrode consumption rate observed at Anglo Platinum's Polokwane smelter. However, when viewed in the context of electrode consumption rates reported for other PGM smelters, the modelled electrode consumption rate is feasible as it exceeds the consumption rate reported for Lonmin's Furnace No. 1.

Figure 4.7 presents the modelled freeboard temperature over 48 hours of operation, as well as the average modelled freeboard temperature. Freeboard temperature values for Anglo Platinum's Polokwane smelter have not been reported in literature, therefore the freeboard temperature values generated by this model is compared to the temperatures reported for Lonmin's Furnace No. 1 by Eksteen et al. (2011).

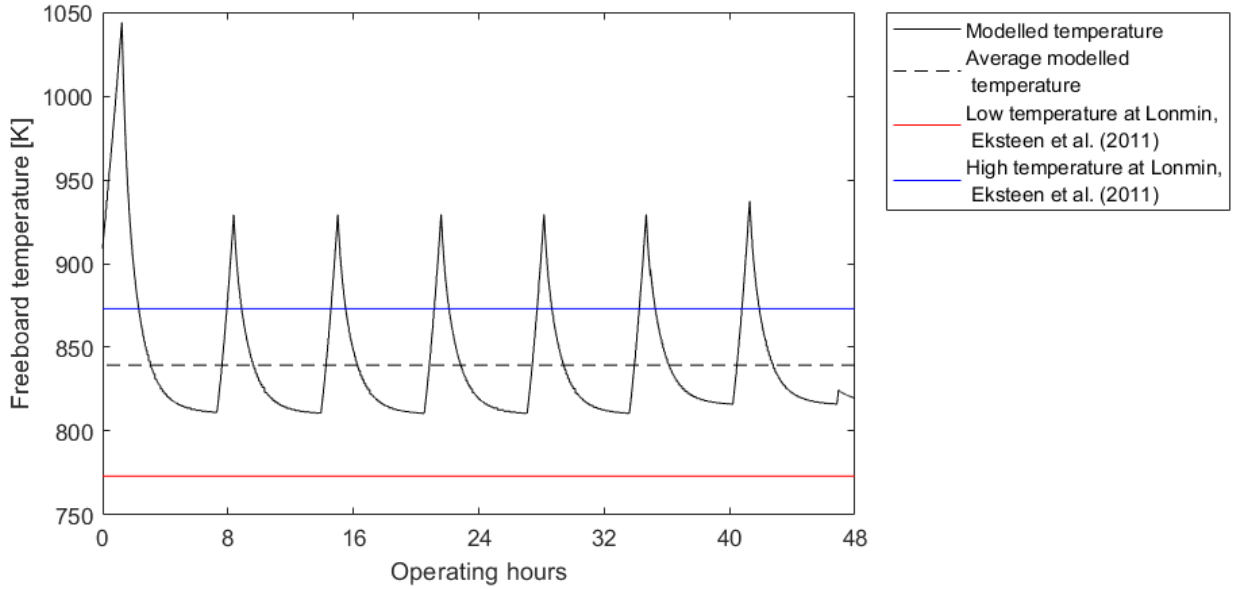

Figure 4.7: Modelled freeboard temperature over two days of simulated operation. The average modelled temperature is indicated by the dashed black line. Minimum- and maximum freeboard temperatures reported by Eksteen et al. (2011) for Lonmin's Furnace No. 1 is indicated by the dashed red- and blue lines, respectively.

Figure 4.7 shows that the modelled freeboard temperature often exceeds the maximum freeboard temperature reported by Eksteen et al. (2011). However, the average modelled freeboard temperature falls between the minimum- and maximum freeboard temperatures reported for Lonmin's Furnace No. 1. Therefore the observed discrepancy is acceptable.

The review of the models by Sheng et al. (1998a, 1998b), Bezuidenhout et al. (2009) and Pan et al. (2011) in section 2.4.4 highlighted that smelting reactions dominate the heat balance over the furnace; most of the energy supplied to the furnace is used to smelt the concentrate while a small fraction is used to raise the temperatures of the distinct zones within the furnace. This model is further validated by confirming that the energy used in smelting reactions is far greater than the energy used to heat zones. Equation 4-35 calculates the energy used for smelting in the smelting concentrate zone:

$$Q_{melt} = F_{melt,\,total}\left(\lambda_C + c_{P,\,C}\left(T_{C,\,melt} - T_{C(S)}\right)\right) \qquad [\,4\text{-}35\,]$$

Using equation 4-35, the smelting reaction energy is calculated from the smelting rate and smelting zone temperature. Q_{melt} is the only reaction that consumes energy over the heat balance of the furnace. Therefore the energy used to heat distinct zones can be calculated as the difference between the energy supplied to the furnace through Joule heating (Q_J, obtained with equation 4-27) and Q_{melt}:

$$Q_{heat} = Q_J - Q_{melt} \qquad [\,4\text{-}36\,]$$

Figure 4.8 illustrates the heat balance over the developed furnace model. The total energy supplied to the furnace, Q_J, the energy consumed in smelting reactions, Q_{melt}, and the energy used for zone heating, Q_{heat}, is plotted over 2 weeks of simulated operation.

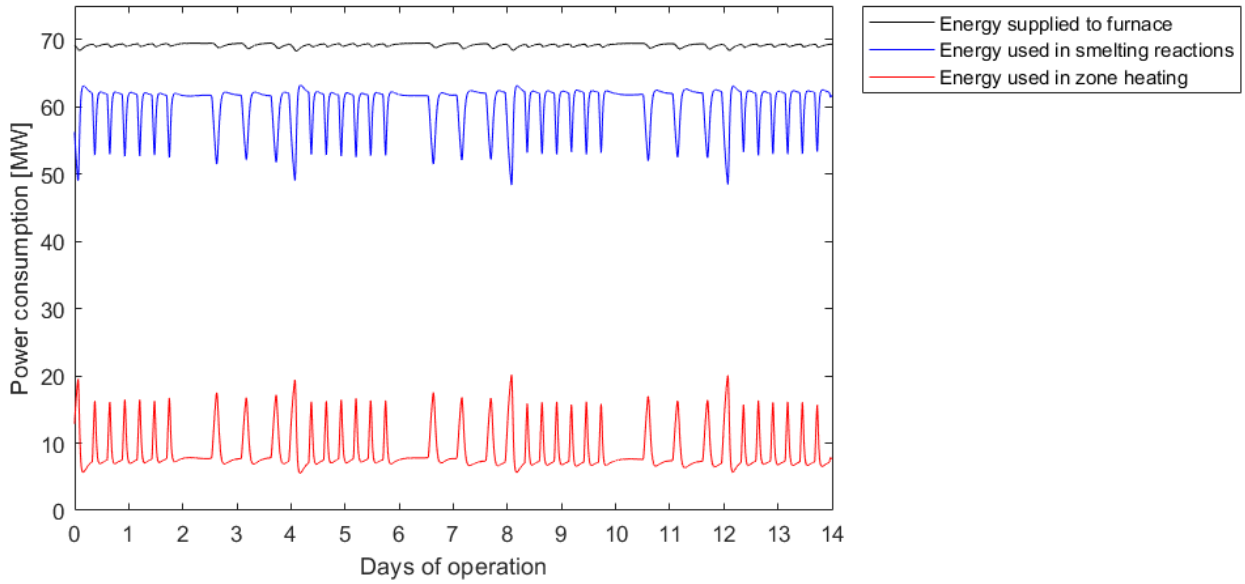

Figure 4.8: Heat balance over the furnace model. The blue line indicates energy used for smelting concentrate, and the red line indicates energy used to heat zones.

Figure 4.8 confirms that most of the energy supplied to the slag zone through Joule heating is consumed in the smelting zone to smelt the concentrate. This indicates that the heat balance of the developed furnace model aligns with heat balances by Sheng et al. (1998a, 1998b), Bezuidenhout et al. (2009) and Pan et al. (2011). The fraction of heat used in smelting reactions over the course of the simulation is calculated with equation 4-37:

$$q_{melt} = \frac{\int_0^{t_{end}} Q_{melt} dt}{\int_0^{t_{end}} Q_J dt} \qquad [4\text{-}37]$$

$q_{melt} = 0.872$ for the presented furnace model. This means that of the energy supplied to the furnace in the presented model, 87.2% is used in smelting reactions.

The model validation is summarized as follows: the developed furnace model generates plausible matte- and slag temperatures. The power supplied to it, the concentrate throughput and the electrode oxidation rate observed in the furnace model aligns with production values observed in Anglo Platinum's Polokwane smelter. The mean freeboard temperature of the developed model falls within the temperature bounds observed at Lonmin's Furnace No. 1. Finally, 87.2% of the energy supplied in the furnace model is used in smelting reactions, confirming that the furnace model aligns with the heat balance performed by Sheng et al. (1998a, 1998b).

4.6 Simulated data

This section describes the simulated data generated by the furnace model. The blowback mechanism is illustrated, and the specifications for the FPR model development dataset obtained from this furnace model are discussed. Finally, faults that are introduced over the course of this simulation are discussed and illustrated.

4.6.1 Furnace blowbacks

The proposed blowback mechanism (equations 4-30 to 4-34) is closely linked to concentrate bed thickness; a thick concentrate bed would release large volumes of reaction gases to the freeboard when ruptures form. This simulation forces blowbacks in the model by changing the concentrate charging rate (equation 4-19) to maintain the bed thickness at higher levels. Figure 4.9 shows how blowbacks are caused by introducing a fault where the bed thickness is maintained at too high levels.

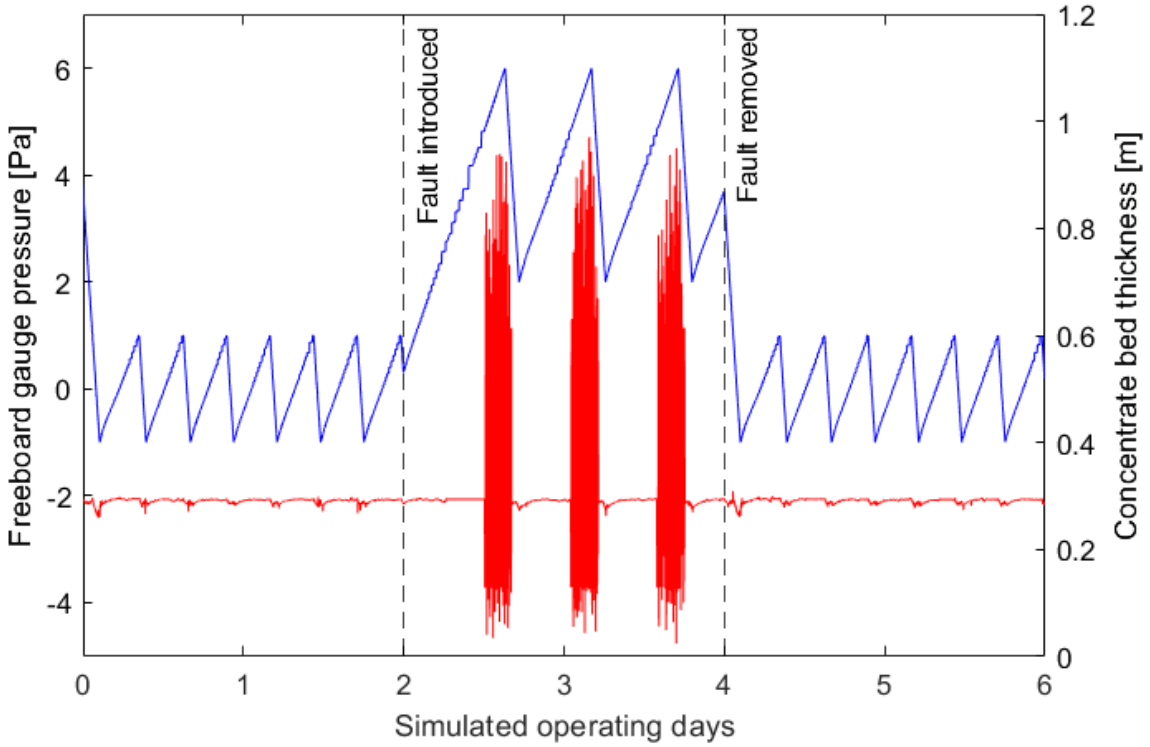

Figure 4.9: Illustration of how increasing the concentrate bed thickness (blue) causes furnace blowbacks, where the freeboard gauge pressure (red) becomes positive. A blowback-causing fault is introduced at 2 days of simulated operation; during this time the bed thickness is maintained at levels where blowbacks occur when the bed ruptures.

Figure 4.9 also shows that concentrate bed thickness should not be included in datasets used for FPR model development. The blowbacks generated by the presented model are easily predicted by simply observing bed thickness, resulting in an overly simple monitoring problem. Therefore bed thickness is omitted from the list of monitored variables presented in the next section.

4.6.2 Monitored variables and case study

Measured variables generated by the ODE model are listed in Table 4.3. Each of the variables given in Table 4.3 can be measured reliable on industrial SAFs. The dataset is constructed by sampling the output of the ODE model at every ten seconds of simulated operation; this is the sampling frequency used by sensors on Anglo Platinum's Polokwane smelter (Groenewald et al., 2018). Note that Table 4.3 omits the concentrate bed thickness.

Table 4.3: Monitored variables in the simulated dataset

Monitored variable		Symbol	Units
1	Slag zone height	L_S	m
2	Matte zone height	L_M	m
3	Slag zone temperature	T_S	K
4	Matte zone temperature	T_M	K
5	Bulk concentrate temperature	$T_{C(B)}$	K
6	Freeboard temperature	T_G	K
7	Cooling water temperature	T_W	K
8	Reaction gas concentration in freeboard	$C_{G,R}$	mol/m^3
9	Freeboard pressure	P_G	Pa

The furnace model is run to simulate 12 weeks of furnace data. This yields a dataset with the specifications given in Table 4.4:

Table 4.4: Case study dataset specifications

Simulation duration	$12\ weeks$	Number of measured variables	9
Sampling rate	$10\ s^{-1}$	Dataset dimensionality	725380×9
Number of samples	725380	Number of blowbacks	63

4.6.3 Faults and disturbances introduced

This model introduces four different faults and disturbances over the course of its three month simulation. Table 4.5 presents a very brief overview of these events, and shows which variables in the derived model given in section 4.3 are changed to facilitate different faults or disturbances.

Table 4.5: Overview of faults and disturbances introduced over the simulation

	Fault or disturbance description	Normal operation	Altered operation
1.	Blowback-causing fault.	$L_{C,\,upper\ lim.} = 0.6\ m$ $L_{C,\,lower\ lim.} = 0.4\ m$	$L_{C,\,upper\ lim.} = 1.1\ m$ $L_{C,\,lower\ lim.} = 0.7\ m$
2.	Cooling water flowrate halved.	$F_W = 2400\frac{mol}{s} = 43.2\frac{L}{s}$	$F_W = 1200\frac{mol}{s} = 21.6\frac{L}{s}$
3.	Extraction draught lowered.	$P_{ext,\,gauge} = -10\ Pa$	$P_{ext,\,gauge} = -8\ Pa$
4.	Charge composition switch from Merensky-UG2 blend to primarily Merensky ore	$x_{charge,\,XO} = 0.90$ $x_{charge,\,XS} = 0.09$ $x_{charge,\,XS_2} = 0.01$	$x_{charge,\,XO} = 0.78$ $x_{charge,\,XS} = 0.21$ $x_{charge,\,XS_2} = 0.01$

Table 4.5 lists two faults and one disturbance in addition to the blowback-causing fault that is paramount to this study. These additional faults and disturbances are not the objective of this study, but they are included in the simulation to test FPR models' ability to distinguish the blowback-causing fault from multiple operating conditions.

Table 4.5 shows that furnace blowbacks are effected by increasing the concentrate bed upper- and lower limits, forcing reaction gas build-up that cause bed ruptures, and hence blowbacks. This is illustrated in Figure 4.10, where the blowback-causing fault is introduced and subsequently removed in two day intervals.

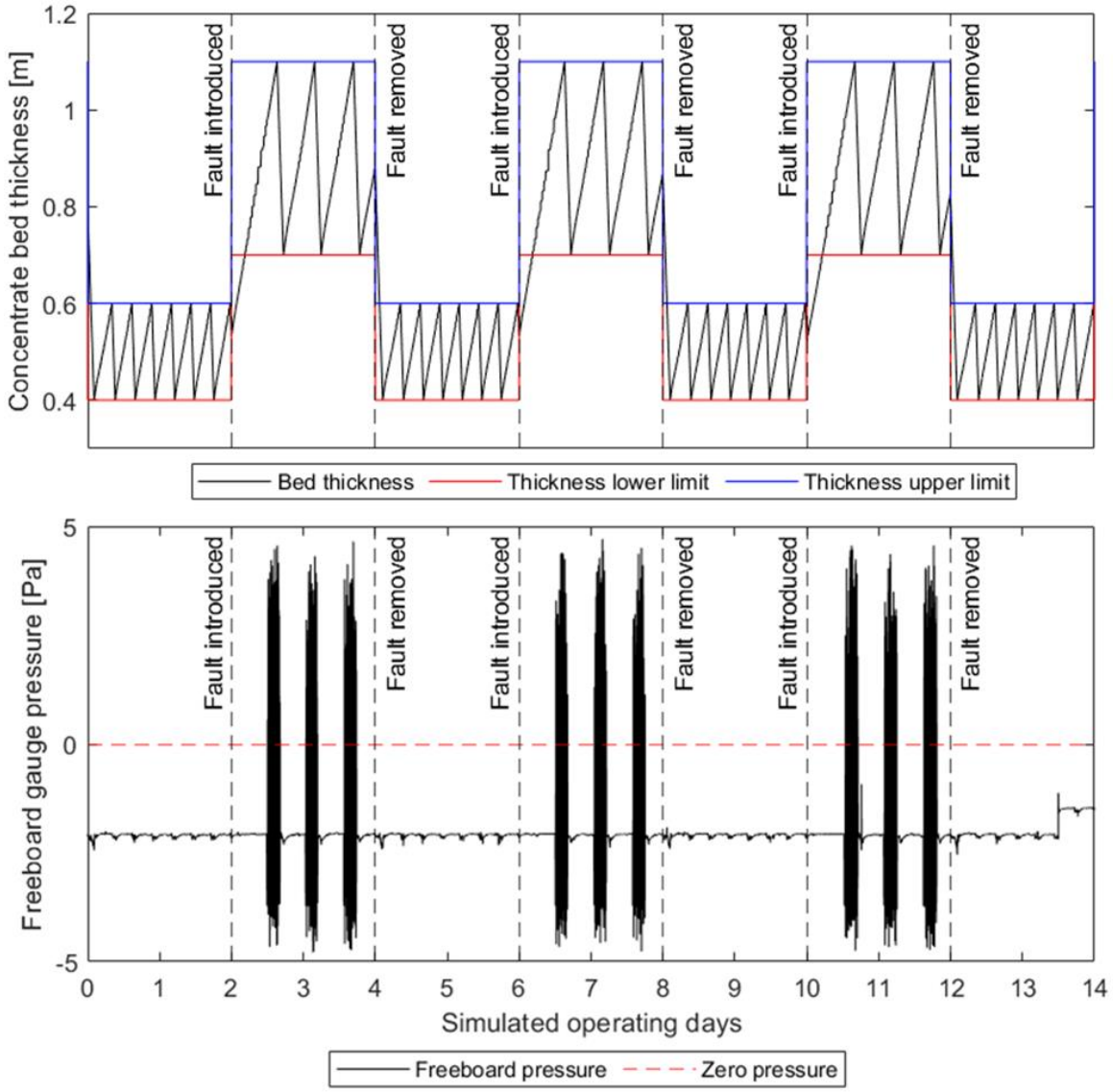

Figure 4.10: Illustration of blowback-causing faults in the simulated furnace data. A fault is introduced by increasing the bed thickness upper- and lower limits (blue- and red lines in the first graph, respectively). The second graph shows how this reliably causes blowbacks to occur, where the freeboard pressure becomes positive.

The second fault shown in Table 4.5 is a drop in cooling water flowrate. This fault is introduced and subsequently removed in three week intervals, and its effect on the cooling water temperature is illustrated in Figure 4.11.

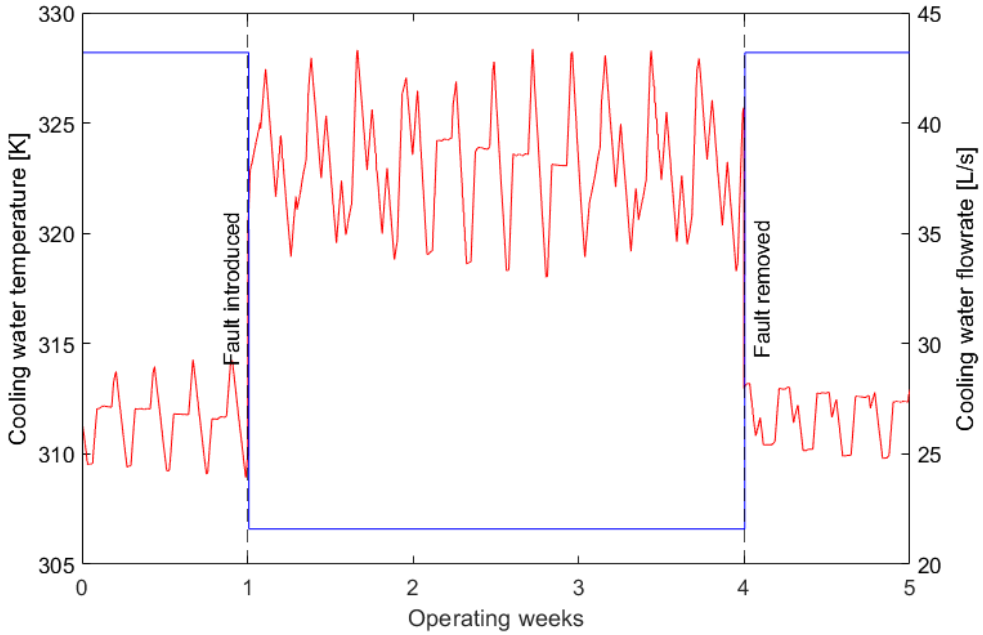

Figure 4.11: Illustration of the cooling water flowrate (blue line) drop and subsequent increase in cooling water temperature (red line).

The third fault given in Table 4.5 is a drop in the extraction draught. This fault is introduced and subsequently removed every two-and-a-half weeks, and its effect on freeboard gauge pressure is illustrated in Figure 4.12.

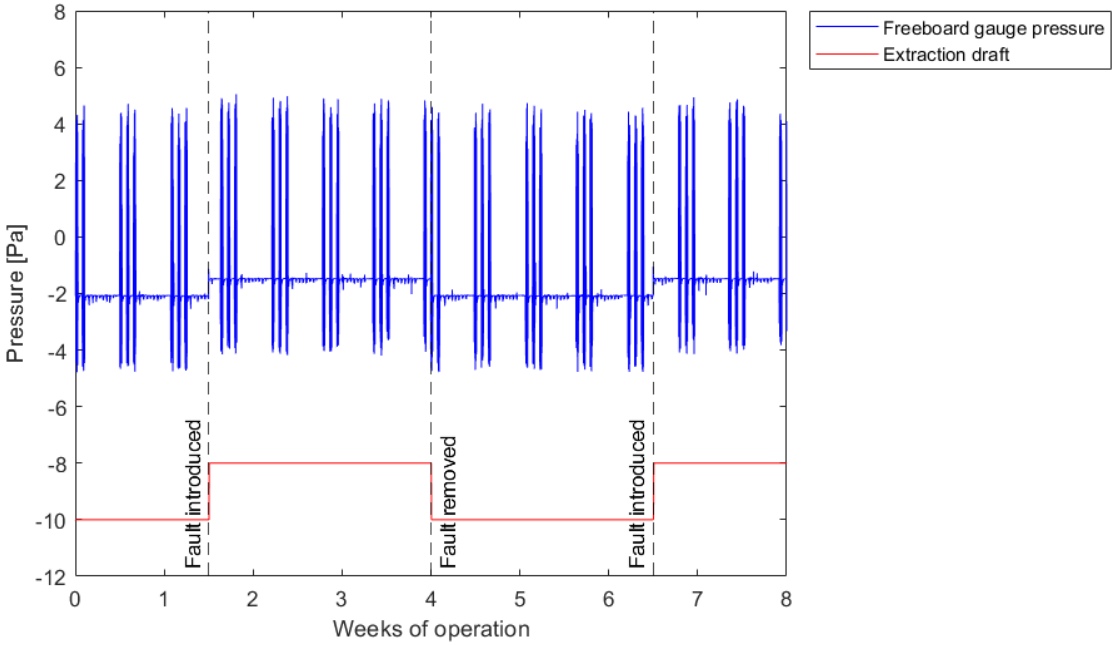

Figure 4.12: Illustration of the drop in extraction draught pressure (red) and its effect on furnace freeboard gauge pressure (blue)

The final fault shown in Table 4.5 reflects a change in charge composition from a feed consisting of a blend of UG2- and Merensky derived ore, to a feed consisting only of Merensky ore. This is simulated by increasing the fraction of sulphide components while decreasing the fraction of oxide- and silicate components in the feed. The effect that this change in composition has on the composition of the bulk concentrate is illustrated in Figure 4.13, where the ore composition switches every week:

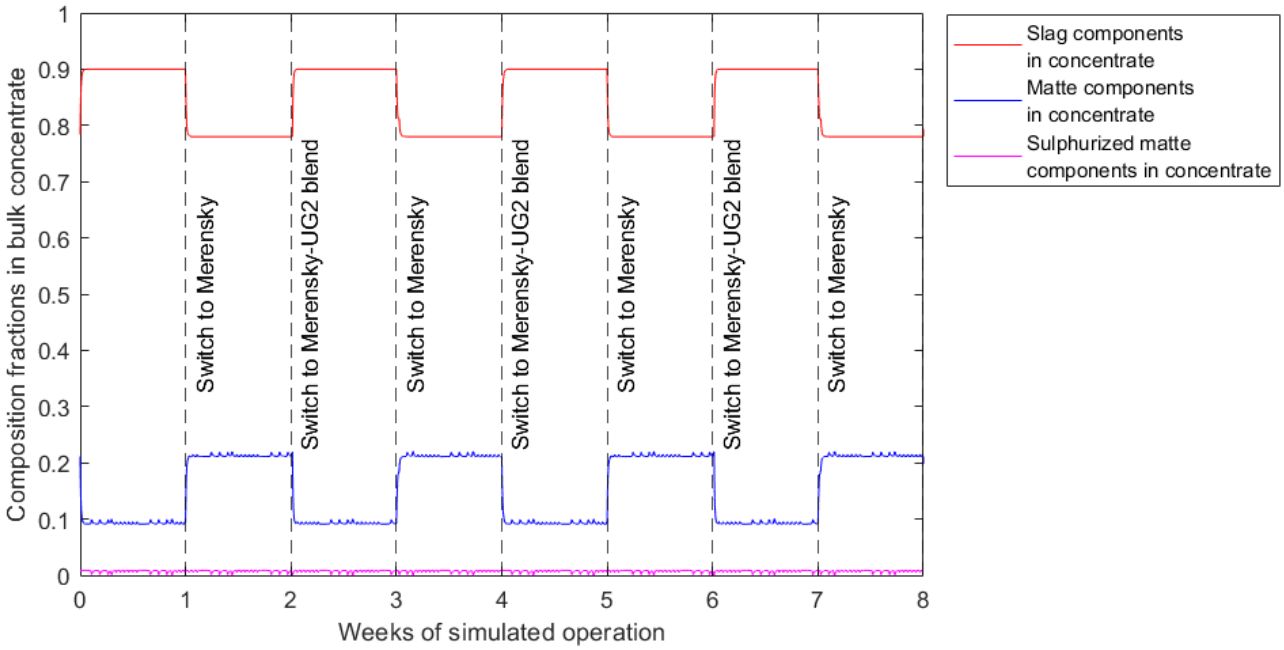

Figure 4.13: Illustration of the effect of switching between a primarily Merensky feed to a Merensky-UG2 feed blend. The Merensky ore causes a sharp increase in the amount of matte components in the concentrate (blue line) while decreasing the amount of slag components in the concentrate (red line). The change in composition did not affect the amount of sulphurized matte (magenta line) in the concentrate.

4.7 Numerical implementation of developed model

The developed model is implemented and solved numerically in MATLAB using *ode15s*. *ode15s* is a stiff differential equation solver that integrates a set of ordinary differential equations over a set time span. Using *ode15s*, equations 4-1 to 4-17 are solved numerically over a span of 12 weeks, yielding sufficient data to evaluate FPR models. A stiff solver was favoured to ensure that the ODE model is integrated successfully.

The MATLAB code used for solving the ODEs of the developed furnace model numerically and generating 12 weeks of simulated furnace data is presented in Appendix D. *ode15s* requires that the initial values of the ODE variables are specified before differential equations can be solved. Table 4.6 presents the ODE variables, their respective differential equations, as well as their initial values assumed for this model.

Table 4.6: Initial values used in numerical integration of the ODEs of the developed model

ODE variable		Symbol	Equation no.	Initial value
1.	Slag compounds in bulk concentrate	$N_{C(B),XO}$	4-1	$7.8 \cdot 10^5\ mol$
2.	Matte compounds in bulk concentrate	$N_{C(B),XS}$	4-2	$5.75 \cdot 10^5\ mol$
3.	Sulphurized matte in bulk concentrate	$N_{C(B),XS_2}$	4-3	$0\ mol$
4.	Bulk concentrate temperature	$T_{C(B)}$	4-4	$1100\ K$
5.	Slag compounds in smelting concentrate	$N_{C(S),XO}$	4-5	$7.8 \cdot 10^5\ mol$
6.	Matte compounds in smelting concentrate	$N_{C(S),XS}$	4-6	$5.75 \cdot 10^5\ mol$
7.	Sulphurized matte in smelting concentrate	$N_{C(S),XS_2}$	4-7	$0\ mol$
8.	Smelting concentrate temperature	$T_{C(S)}$	4-8	$1400\ K$
9.	Reaction gas in concentrate	$N_{C(R)}$	4-9	$0\ mol$
10.	Moles of liquid slag	N_S	4-10	$1.1 \cdot 10^7\ mol$
11.	Slag zone temperature	T_S	4-11	$1900\ K$
12.	Moles of liquid matte	N_M	4-12	$6.5 \cdot 10^6\ mol$
13.	Matte zone temperature	T_M	4-13	$1750\ K$
14.	Air components in freeboard	$N_{G,A}$	4-14	$1830\ mol$
15.	Reaction gas components in freeboard	$N_{G,R}$	4-15	$0\ mol$
16.	Freeboard temperature	T_G	4-16	$900\ K$
17.	Cooling water temperature	T_W	4-17	$300\ K$

The initial slag, matte, bulk- and smelting concentrate temperatures given in Table 4.6 were selected as the steady state SAF values given by Eksteen (2011). The initial freeboard temperature is the steady state freeboard temperature in the simulator by Pan et al. (2011). The total amounts of matte- and slag components present at the start of the simulation is obtained from the throughput of the Polokwane furnace and its mean residence time reported by Crundwell et al. (2011a).

5 FAULT PATTERN RECOGNITION APPROACH

The chapter presents the combined FPR approach used in this project, and illustrates how the second- and third objectives identified for this project are met. Section 5.1 describes which feature engineering techniques are applied to the simulated data generated by the ODE model, and presents the relevant equations. Section 5.2 shows how the engineered dataset is partitioned into target- and testing data. Section 5.3 describes how reconstruction errors generated by the developed models will be evaluated and how their discriminants will be presented. Section 5.4 presents the model evaluation equations used to express derived model performance. Sections 5.5 to 5.8 shows the algorithms used for developing- and applying PCA, kernel PCA, AE- and CAE models, respectively.

Figure 5.1 gives a structured overview of the FPR approach used, and guides the reader to where the equations/algorithms presented in this chapter falls in the overall approach to FPR used.

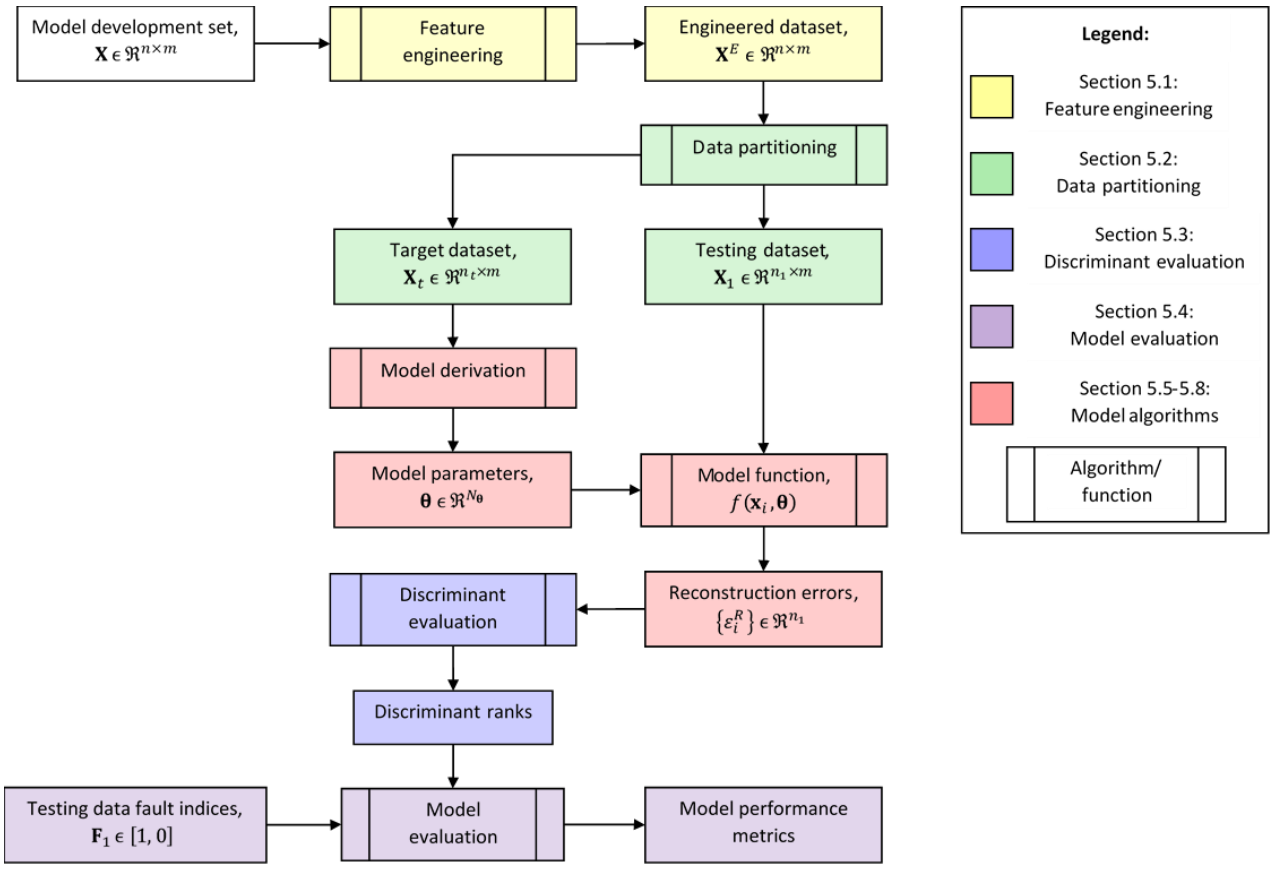

Figure 5.1: Structured FPR approach overview

5.1 Engineered features

Signal de-trending is noted as a prominent feature engineering technique in section 3.2.3; it will allow irrelevant, low frequency features to be removed from $\mathbf{X}$. Kubben et al. (2019) showed that a moving window average is a type of low pass filter; it extracts long-term trends from a signal. By subtracting the moving window average from a signal, long term trends are removed:

$$x_i^E = x_i - \frac{1}{n_1}\sum_{j=i-n_1+1}^{j=i} x_j \qquad [5\text{-}1]$$

x_i^E is the de-trended feature of x_i. The second term computes the mean value of the signal over a window length n_1. As a high pass filter, de-trending does not significantly reduce the signal variance that reconstruction-based FPR models are so susceptible to. Moving window standard deviation, presented in equation 5-2, calculates the standard deviation of groups of signal values.

$$x_i^E = \sqrt{\frac{1}{n_2-1}\sum_{j=i-n_2+1}^{j=i}\left(x_i - \frac{1}{n_2}\sum_{k=i-n_2+1}^{k=i} x_k\right)^2} \qquad [5\text{-}2]$$

x_i^E is the moving standard deviation of x calculated over a window size n_2. Figure 5.2 illustrates the feature obtained by equations 5-1 and 5-2 from the simulated freeboard pressure signal:

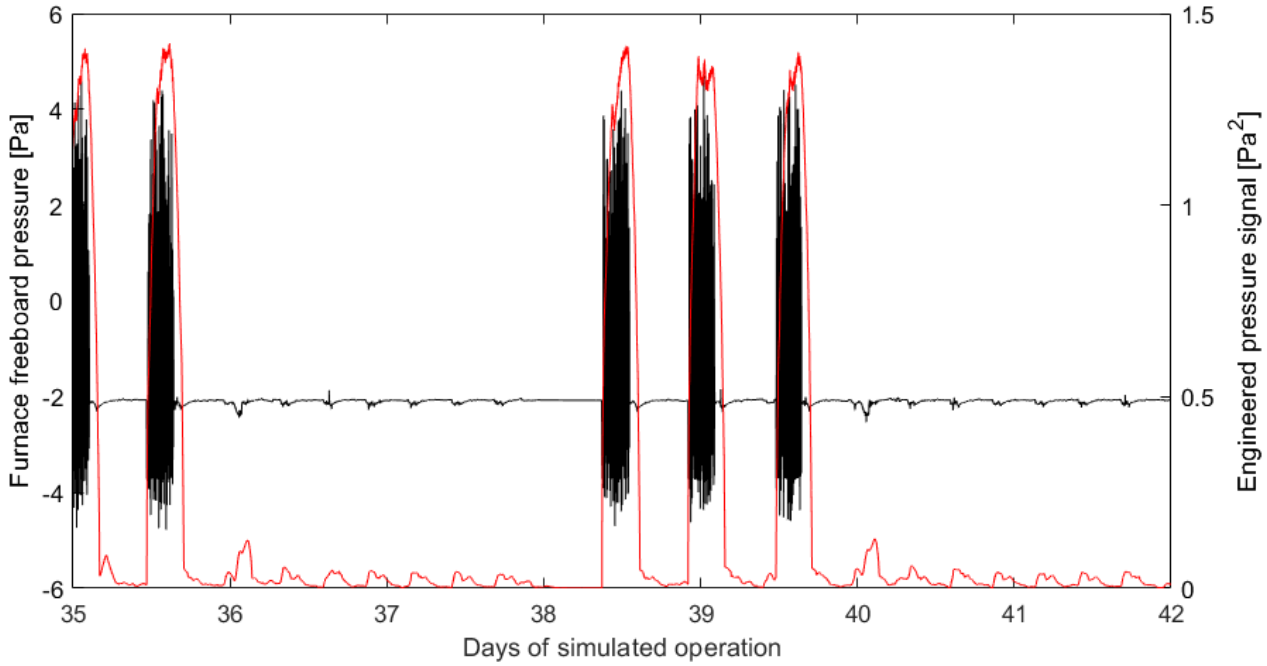

Figure 5.2: Illustration of the original- (black) and engineered- (red) SAF freeboard pressure signal. The moving mean freeboard pressure is calculated in a window with a length $n_1 = 2160$, and subtracted from the pressure signal. The moving standard deviation is calculated for the resulting de-trended signal for a window with length $n_2 = 570$.

Table 5.1 shows which feature engineering techniques are applied to which monitored variables. Moving window standard deviations are applied to high-variance signals, and all signals are de-trended.

Table 5.1: Feature engineering techniques applied to simulated data

Monitored variable		FE technique(s)	FE parameters
1	Slag zone height, L_S	Moving mean de-trending	$n_{1,L_S} = 1560$
2	Matte zone height, L_M	Moving mean de-trending	$n_{1,L_M} = 2490$
3	Slag zone temperature, T_S	Moving mean de-trending	$n_{1,T_S} = 2100$
4	Matte zone temperature, T_M	Moving mean de-trending	$n_{1,T_M} = 2700$
5	Bulk concentrate temperature, $T_{C(B)}$	Moving mean de-trending	$n_{1,T_{C(B)}} = 1020$

6	Freeboard temperature, T_G	Moving mean de-trending Moving standard deviation	$n_{1,T_G} = 960$ $n_{2,T_G} = 570$
7	Cooling water temperature, T_W	Moving mean de-trending Moving standard deviation	$n_{1,T_W} = 1230$ $n_{2,T_W} = 630$
8	Reaction gas concentration in freeboard, $C_{G,R}$	Moving mean de-trending Moving standard deviation	$n_{1,C_{G,R}} = 1110$ $n_{2,C_{G,R}} = 660$
9	Freeboard pressure, P_G	Moving mean de-trending Moving standard deviation	$n_{1,P_G} = 2160$ $n_{2,P_G} = 570$

The window length values given in Table 5.1 were obtained by optimizing the window lengths to maximize the validation performance of a simple PCA model trained with training data. Moving window standard deviation is applied to the signals with the highest variance: freeboard temperature, cooling water temperature, reaction gas concentration in the freeboard and freeboard pressure.

Data standardization is noted as a common feature engineering technique in section 3.2.3. Standardization constants are obtained from the target dataset, $\mathbf{X}_t$, by applying equations 5-3 and 5-4:

$$\mu_j = \frac{1}{n_t}\sum_{i=1}^{n_t} x_{j,i} \qquad [5\text{-}3]$$

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{n_t}(x_{j,i}-\mu_j)^2}{n_t}} \qquad [5\text{-}4]$$

μ_j and σ_j are standardization constants that scale variable j in observations using equation 5-5. Note that these standardization constants are specific to observations from $\mathbf{X}_t$; if patterns in $\mathbf{X}_t$ are characterized by unique means and low standard deviations, then applying equation 5-5 will improve pattern recognition:

$$z_{j,i} = \frac{x_{j,i}-\mu_j}{\sigma_j} \qquad [5\text{-}5]$$

The standardization constants, unlike the de-trending and moving window standard deviation constants, are obtained from $\mathbf{X}_t$; they are functionally similar to model parameters and are treated as such in the modelling algorithms presented in sections 5.5 to 5.8.

5.2 Data partitioning

The importance of testing an FPR model on different data than what was used to train- and optimize the model was highlighted in section 3.2.1. The simulated data generated by the ODE model presented in chapter 4 ($\mathbf{X} \in \Re^{n\times m}$) will therefore be partitioned into training- ($\mathbf{X}_0 \in \Re^{n_0\times m}$), validation- ($\mathbf{X}_1 \in \Re^{n_1\times m}$) and testing ($\mathbf{X}_2 \in \Re^{n_2\times m}$) datasets. This partitioning is shown for the generated freeboard pressure data in Figure 5.3, where 12 weeks of simulated operation is partitioned into training (green)-, validation- (blue) and testing (red) datasets.

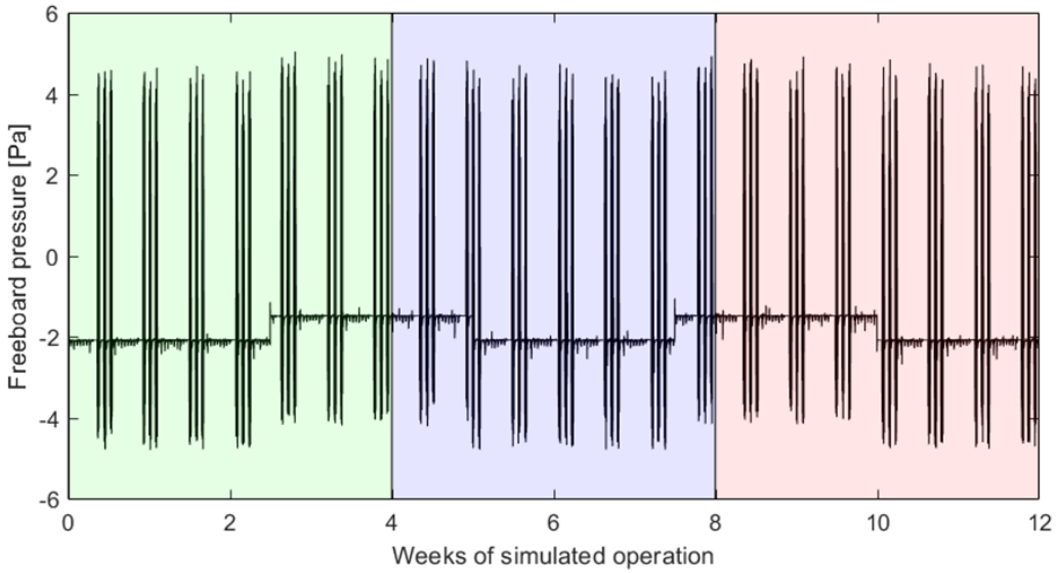

Figure 5.3: Illustration of how the simulated process dataset is partitioned into separate training- (green area), validation- (blue area) and testing- (red area) datasets. Note that all variables in $\mathbf{X}$ are partitioned in the same way, not just the freeboard pressure.

In section 3.2.2 it was noted that reconstruction-based FPR models should only be trained on faulty data. Therefore a target dataset, $\mathbf{X}_t \in \Re^{n_t \times m}$, is constructed as a subset of $\mathbf{X}_0$. These target observations have to be selected in a way that could also be applied to an industrial, ill-defined dataset. Figure 5.4 illustrates the ground truth of the model development dataset, $\mathbf{X}$. It shows where the blowback-causing fault is absent in $\mathbf{X}$, and hence where recognitions by the developed models would be true positives.

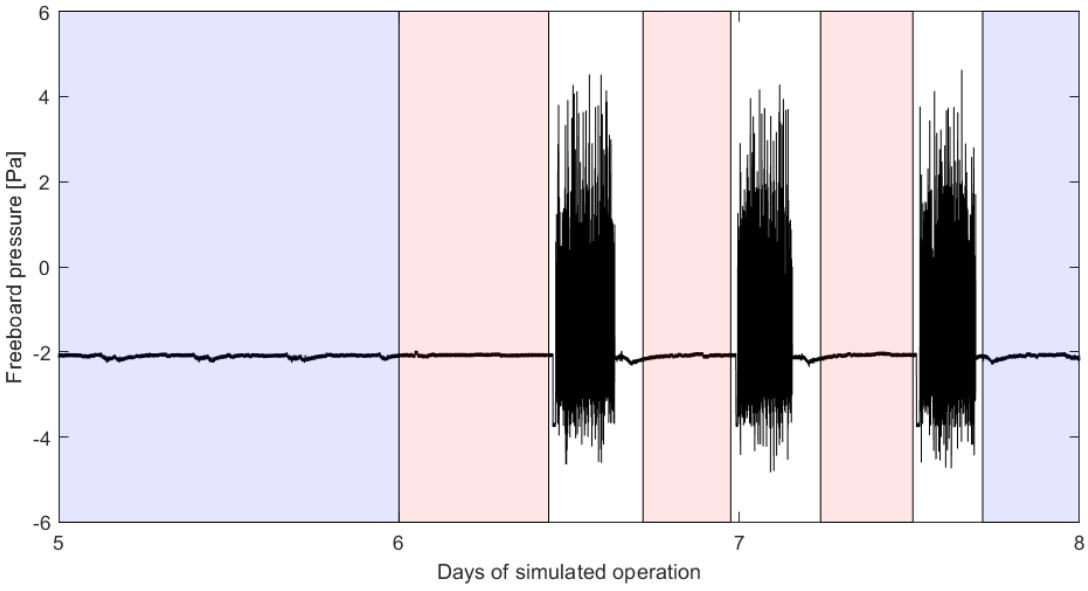

Figure 5.4: Illustration of the ground truth of the simulated data w.r.t. the presence of blowback-causing faults. Fault-free observations are found in the area shaded blue. Faulty observations are found in the area shaded red. Unshaded areas contain blowback-causing faults, but sounding the alarm here would either be redundant, due to blowbacks already occurring, or would not provide sufficient warning before the blowback.

Most industrial processes (as well as the simulated process considered here) are fault tolerant; a fault condition is present for some time until a critical failure occurs (Salfner et al., 2010). Therefore a blowback prediction is assumed to be valid for a time period ($\Delta t_{prediction}$). The prediction is correct if a failure occurs in this window, and if enough time ($\Delta t_{warning}$) is available to operators to take corrective measures. These metrics are illustrated in Figure 5.5:

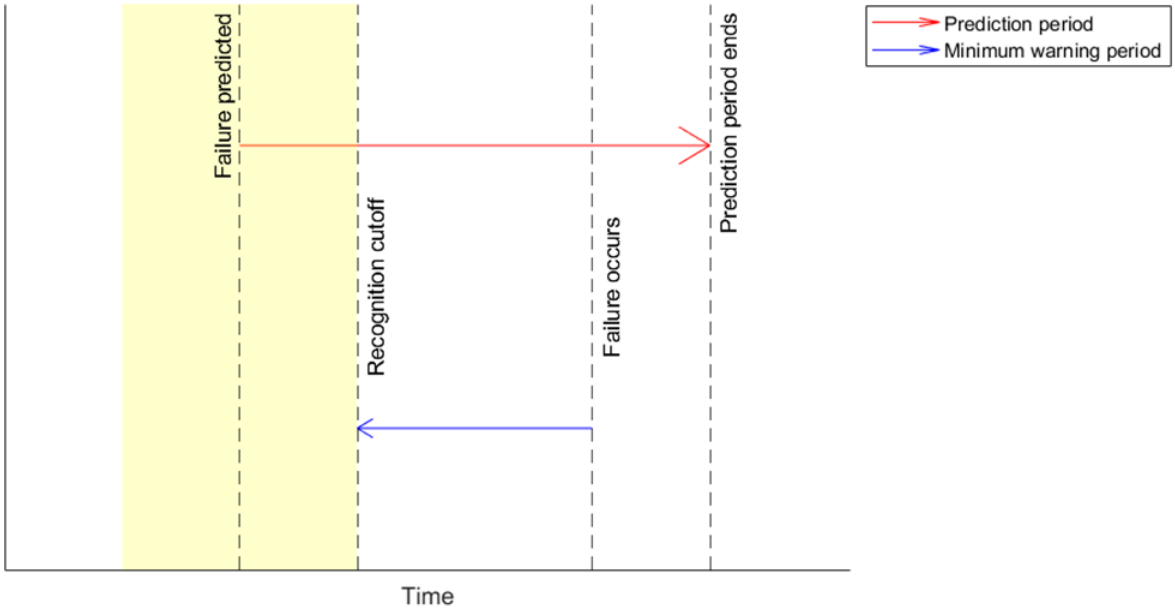

Figure 5.5: Illustration of online fault recognition. After a failure is predicted, a failure is assumed to occur within the prediction period (red arrow). The prediction is valid if a failure occurs within this period. The prediction should provide a minimum warning period (blue arrow) for plant operators to prepare for the failure. Therefore, given the prediction- and minimum warning periods, only warnings given in the gold shaded area will be both valid and provide plant operators with sufficient time to prepare for the failure.

Specifying $\Delta t_{prediction}$ and $\Delta t_{warning}$ allows a window before each blowback, where blowback predictions would be valid, to be defined. Figure 5.6 illustrates observations in $\mathbf{X}_0$, highlighted in gold that are selected as target observations for $\Delta t_{prediction} = 1.5\ h$ and $\Delta t_{warning} = 0.5\ h$. These observations are selected from the training data (as partitioned in Figure 5.3) to construct the target dataset, $\mathbf{X}_t$.

Note that $\mathbf{X}_t$ does not include all faulty observations; the above approach is simply a way of constructing $\mathbf{X}_t$ with observations with characteristics that, if recognized, will flag observations preceding each blowback. Recognitions succeeding these windows will not reduce model specificity; blowback-preceding conditions are present and therefore an alarm will not be a false positive. However, they do not improve sensitivity; they do not provide enough of a warning to allow for corrective actions. However, recognitions preceding these windows are true, and a good reconstruction-based FPR model would generate high discriminant statistics for these observations.

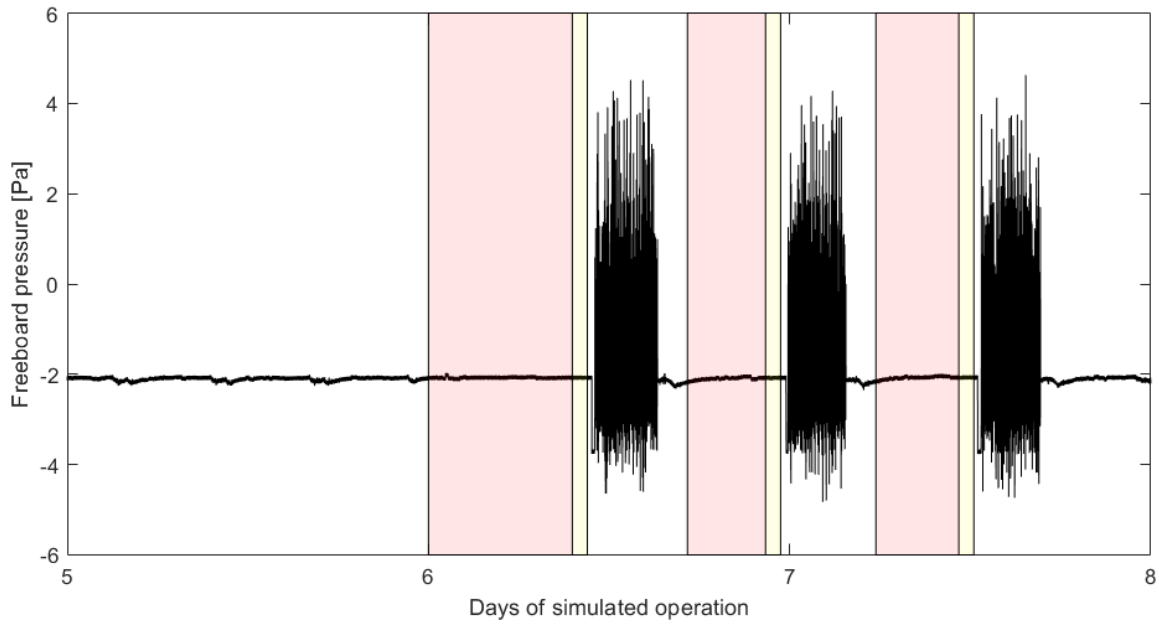

Figure 5.6: Illustration of observations selected for the target dataset, $\mathbf{X}_t$. $\mathbf{X}_t$ is constructed from observations in the gold-shaded region, but fault conditions are still present outside this window in the red shaded area.

5.3 Discriminant evaluation and presentation

This section describes how discriminant values are evaluated and presented. A discriminant value, u_i, for observation $\boldsymbol{x}_i$, is calculated as the inverse of the reconstruction error (this is presented in equations 3-2 to 3-4). The discriminant value, u_i, is compared to a recognition threshold, τ, yielding a logical output. If u_i is above τ, then the FPR model flags $\mathbf{x}_i$:

$$q_i = \begin{cases} 1 \; if \; u_i \geq \tau \\ 0 \; if \; u_i < \tau \end{cases} \qquad [5\text{-}6]$$

If discriminant values are evaluated in isolation, then the logistical output from equation 5-6 will classify the patterns (as in equation 3-5). As discussed in section 3.7.2, the generated discriminant values, u_i, are very noisy and FPR based on single u_i-values will be unreliable. FPR is therefore performed on a window of discriminant values; if the number of flagged values exceed a critical threshold, then the observation $\mathbf{x}_i$ is recognized by the model:

$$\gamma_i = \begin{cases} C_1 \; if \; \left(\sum_{i-l_w}^{i} q_i\right) \geq r_\alpha \\ C_0 \; if \; \left(\sum_{i-l_w}^{i} q_i\right) < r_\alpha \end{cases} \qquad [5\text{-}7]$$

r_α is the number of flagged observations needed from l_w observations to recognize a faulty pattern with α confidence. r_α is calculated from a binomial distribution using the approach by Singhal and Seborg (2002, 2000), which is explored in section 3.7.2. This thesis assumes that each discriminant has a 75 % probability ($\beta = 0.75$) to exceed a given recognition threshold, and recognitions are made at 90 % confidence ($\alpha = 0.90$). Table 5.2 presents which discriminant evaluation parameters are considered in this project.

Table 5.2: Discriminant evaluation parameters

Configuration parameters	Symbol	Investigated values
Assumed discriminant probability	β	0.75
Recognition confidence	α	0.9
Recognition window lengths	l_w	$0, 30, 60, \ldots, 330, 360$

As discussed in section 3.2.2, recognition thresholds of reconstruction-based FPR models are defined empirically on generated discriminant values. Furthermore a single large discriminant may not lead to a recognition if other discriminant values in its recognition window are low. Consequently a discriminant's "strength" – how strongly it suggests that a fault is occurring – is not necessarily expressed by its value, and presenting these values will give confusing representations of FPR performance.

The strength of a discriminant value is better expressed by that value's rank; a discriminant's rank shows at which percentile of the generated discriminant values a fault is indicated. Discriminant ranks provide more meaningful representations of a model's recognition output, and are illustrated in Figure 5.7:

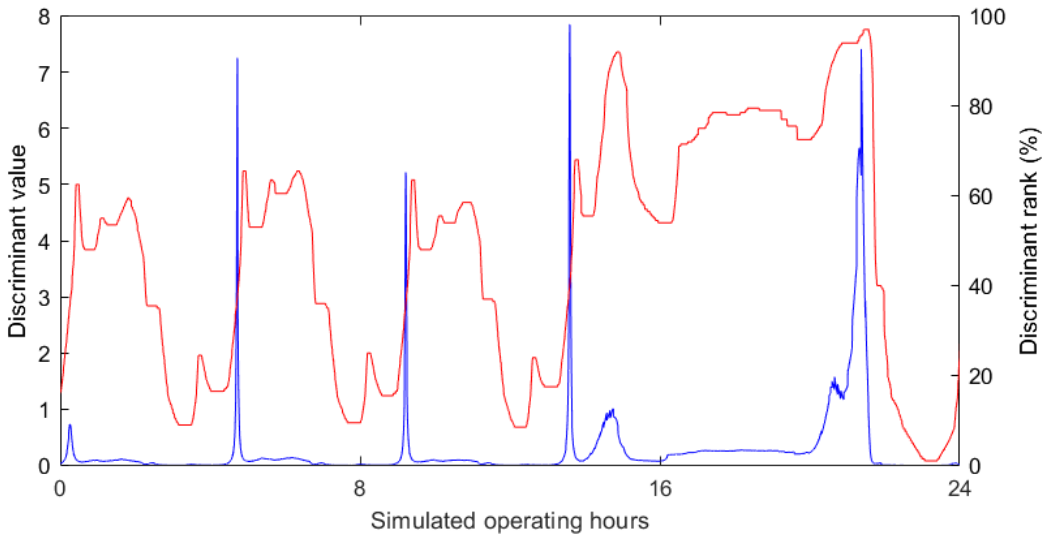

Figure 5.7: Discriminant recognition ranks (red) obtained from discriminant values (blue).

Figure 5.7 shows the discriminant ranks obtained after applying moving window recognition ($l_w = 135$; $r_{\alpha\ =0.9} = 108$) to generated discriminant values. Note how the discriminant rank effectively captures at which thresholds each observations will be recognized; individual high value discriminants values are likely outliers and are ranked lower than discriminants preceded by many moderately-high values.

5.4 Model evaluation

As discussed in section 3.7.1, the area under the ROC-curve (AUC_{ROC}) is a useful metric that expresses the likelihood that a faulty observation has a higher discriminant value than a fault-free observation, and is uniquely suited for expressing reconstruction-based FPR model performance. However, AUC_{ROC} should not be evaluated over all possible recognition thresholds, as this inflates model performance. This section describes how minimum (δ_1) and maximum (δ_2) specificities are defined.

ROC-curves effectively capture specificity (δ)- and sensitivity (φ) in one graph, and AUC_{ROC} combines these metrics into one value. However, precision (ψ) is a function of δ, φ, number of faulty (F_1)- and non-faulty (F_0) observations in the test dataset. Equation 5-8 reorganizes the formal definition of precision given in Table 3.2 to be a function of the aforementioned terms:

$$\psi = \frac{\varphi}{\varphi + \frac{F_0}{F_1}(1-\delta)} \qquad [5\text{-}8]$$

Equation 5-8 shows that a high specificity (δ) and sensitivity (φ) does not necessarily guarantee good model performance in datasets where the number of fault-free observations, F_0, is more than the number of faulty observations, F_1. Unfortunately, this is the case in the simulated data used as well as on industrial plants, therefore the model will be evaluated over specificities where a minimum precision can be met. Sensitivities (φ) higher than 100% are impossible, therefore the lowest specificity (δ_1) where a minimum precision (ψ_{min}) can be achieved is defined by equation 5-9:

$$\delta_{\psi_{min}} = 1 + \frac{F_1}{F_0}\left(\frac{\psi_{min}-1}{\psi_{min}}\right) \qquad [5\text{-}9]$$

$\delta_{\psi_{min}}$ is the specificity where the model sensitivity should be 100% to reach a minimum precision ψ_{min}. An upper specificity (δ_2) need not be defined, as a model that is 100% specific can theoretically meet both sensitivity and precision requirements. Equation 5-10 shows how FPR model performance will be quantified to account for precision:

$$AUC_{ROC}\left(\delta_{\psi_{min}}, 1\right) = \frac{\int_{\delta_{\psi_{min}}}^{1} \varphi \, d\delta}{\left(1-\delta_{\psi_{min}}\right)} = AUC_{ROC,\,\psi_{min}} \qquad [5\text{-}10]$$

Equation 5-10 is a modification of the partial AUC_{ROC} equation presented by Dodd and Pepe (2003); it includes a denominator term of the total area in the ROC-chart above the minimum specificity, $\delta_{\psi_{min}}$. This modification is made to improve the interpretability of the AUC_{ROC}-value. Using this modification, $AUC_{ROC,\,\psi_{min}}$ expresses the fraction of the ROC chart area where the minimum precision, ψ_{min}, can be achieved that falls below the ROC curve.

$AUC_{ROC,\,\psi_{min}}$ is useful for condensing different aspects of FPR performance, like sensitivity, specificity and precision into a single metric directly from discriminant values. This allows different models to be compared objectively and optimal model configurations to be identified quickly.

However, the $AUC_{ROC,\,\psi_{min}}$-value aggregates sensitivity over specificities where a minimum precision can be achieved. Therefore a selected model should still be evaluated on FPR performance metrics obtained for defined recognition thresholds. This project uses simulation data for which the ground truth is known. Therefore, an optimal model's detection delay can also be evaluated. Detection delay is defined as the difference between the time where a fault occurs and the time where it is detected:

$$\Delta t_{DD} = t_{detection} - t_{fault} \qquad [5\text{-}11]$$

5.5 Principal component analysis

This section presents the algorithms used for deriving PCA parameters from a target dataset, $\mathbf{X}_t$, and applying them to calculate reconstruction error for new observations as part of an FPR-model. First an overview of the model is given, showing which model-specific design parameters are defined before model parameters are derived. Then the algorithm for deriving those model parameters is presented. Finally, the algorithm for applying the derived model for FPR is presented. This section concludes by expanding on how the presented algorithms were implemented numerically in MATLAB.

Table 5.3 presents the PCA design- and model parameters. It also shows which different values of the design parameters will be investigated.

Table 5.3: PCA design- and model parameters

Design parameter	**Symbol**	**Investigated values**
Number of retained components	$v \in \mathbb{N}$	$1, 2, 3, \dots, 8$
Lag dimension	$l \in \mathbb{N}_0$	$0, 1, 2, \dots, 10$

Model parameters:			
Target data means	$\boldsymbol{\mu}_t \in \Re^m$	Target data standard deviations	$\boldsymbol{\sigma}_t \in \Re^m$
Retained principal components	$\mathbf{V} \in \Re^{m(l+1)\times v}$		

Note how few design parameters in Table 5.3 need to be defined to build a PCA model on $\mathbf{X}_t$. This highlights simplicity as a key strength of PCA algorithms.

Table 5.4 presents the PCA model parameter derivation algorithm. The algorithm is applied on a dataset of target observations, $\mathbf{X}_t \in \Re^{n_t \times m}$, with n_t observations of m variables. The third column shows the output from each step in the PCA algorithm. The fourth column shows the equations used in each step. The shaded areas in Table 5.4 indicate model parameters used when applying the PCA model for FPR.

Table 5.4: PCA model parameter derivation algorithm

Step description		**Outputs**	**Equations**
1.	Obtain standardization constants from $\mathbf{X}_t$.	$\boldsymbol{\mu}_t \in \Re^m$ $\boldsymbol{\sigma}_t \in \Re^m$	5-3; 5-4
2.	Standardize $\mathbf{X}_t$ with $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$.	$\mathbf{Z}_t \in \Re^{n_t \times m}$	5-5
3.	Lag the standardized dataset $\mathbf{Z}_t$.	$\mathbf{Z}_t^L \in \Re^{n_t \times m(l+1)}$	3-10
4.	Calculate the eigenvectors- and eigenvalues of the correlation matrix of $\mathbf{Z}_t^L$	$\mathbf{P} \in \Re^{m(l+1)\times m(l+1)}$ $\boldsymbol{\lambda} \in \Re^{m(l+1)}$	3-7
6.	Construct the linear subspace by selecting the v most significant principal components, $\mathbf{V}$.	$\mathbf{V} \in \Re^{m(l+1)\times v}$	Selected from $\mathbf{P}$

The PCA reconstruction error algorithm is presented in Table 5.5. The presented algorithm is for a new observation, $\mathbf{x}_i \in \Re^m$, with m variables. The second step in the algorithm assumes that observations preceding $\mathbf{x}_i$ are available.

Table 5.5: PCA reconstruction error algorithm

Step description		Outputs	Equations
1.	Standardize $\mathbf{x}_i$, yielding $\mathbf{z}_i$.	$\mathbf{z}_i \in \Re^m$	5-5
2.	Lag $\mathbf{z}_i$ with l observations.	$\mathbf{z}_i^L \in \Re^{m(l+1)}$	3-10
3.	Reconstruct $\mathbf{z}_i^L$ with $\mathbf{V}$.	$\hat{\mathbf{z}}_i^L \in \Re^{m(l+1)}$	3-9
4.	Calculate the reconstruction error.	$\varepsilon_i^R \in \Re$	3-3

The algorithms presented in Table 5.4 and Table 5.5 were implemented in MATLAB. No built-in functions were required for implementing these algorithms, with the exception of the eigenvalue- and eigenvector decomposition step in Table 5.4 (step 4). This decomposition was performed using *eig*. The MATLAB functions used to implement the above algorithms are presented in section E.5 in Appendix E.

5.6 Kernel principal component analysis

This section presents the algorithms for deriving and applying kernel PCA FPR models. It follows the same structure as section 5.5; an overview of design- and model parameters is presented, the algorithm for deriving model parameters is shown and the algorithm for applying the derived FPR model on a new observation is given. Finally, the numerical implementation of the presented algorithms in MATLAB is discussed.

Table 5.6 presents the kernel PCA design- and model parameters. Note that only a single value is investigated for the number of centroid observations, k. Increasing k will increase the approximation quality of the kernel matrix and exponentially increase the computation costs of the presented algorithms. Specifically, the computation cost of eigenvalue- and vector decomposition executed as part of the model parameter derivation algorithm grows cubically ($O(k^3)$) with the number of representative centroids selected.

Table 5.6: Kernel PCA design- and model parameters

Design parameter	**Symbol**	**Investigated values**
Number of retained components	$v \in \mathbb{N}$	$1, 2, \dots, 10$
Choice of kernel function	$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)$	Gaussian, linear
Lag dimension	$l \in \mathbb{N}_0$	$0, 1, 2, \dots, 6$
Number of centroids	$k \in [1, \dots, n_t]$	500

Model parameters:			
Target data means	$\boldsymbol{\mu}_t \in \Re^m$	Target data standard deviations	$\boldsymbol{\sigma}_t \in \Re^m$
Centroid dataset	$\mathbf{L} \in \Re^{k \times m(l+1)}$	Kernel matrix	$\mathbf{K} \in \Re^{k \times k}$
Retained components	$\mathbf{A_V} \in \Re^{k \times v}$	Nonzero components	$\mathbf{A_P} \in \Re^{k \times p}$
Optimal kernel width	$b_{optimal} \in \Re_{>0}$		

The literature review on kernel PCA presented in section 3.4 omitted equations that are not needed to understand kernel PCA, but have to be applied in the kernel PCA algorithms presented in in Table 5.7 and Table 5.8. These omitted equations are marked with an asterisk, and are described below. The k-means cluster algorithm is used to find a centroid dataset with which to approximate the full kernel matrix:

$$\{\mathbf{l}_j\} = \underset{y_1,\dots,y_n,\mathbf{l}_1,\dots,\mathbf{l}_k}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{y_i=j} \left\| \mathbf{z}_i^L - \mathbf{l}_j \right\|^2 \qquad [5\text{-}12]$$

$\mathbf{l}_j$ is a centroid observation in $\mathbf{Z}_t^L$. Each observation in $\mathbf{Z}_t^L$, $\mathbf{z}_i^L$, is assigned to the centroid, $\mathbf{l}_j$ nearest to it. $\mathbf{l}_j$ is then updated to be the mean of the observations assigned to its cluster. The centroid dataset, $\mathbf{L}$, is constructed iteratively between these two steps. Each eigenvector obtained from $\mathbf{K}_c$, $\mathbf{a}_j$, should be a linear combination of a principal component in the mapped feature space, $\mathbf{v}_j$. Principal components, even implicitly mapped ones, are unit vectors, and $\mathbf{a}_j$ is scaled to ensure it represents a unit vector component:

$$\mathbf{a}_j' = \frac{\mathbf{a}_j}{\sqrt{\lambda_j}} \qquad [5\text{-}13]$$

The kernel PCA algorithms presented here use low-rank approximations of the kernel matrix, $\mathbf{K}$, obtained through k-means clustering. A low-rank approximation is never as good as the original (Schölkopf et al., 1998b), therefore performance is lost. The linear kernel function (equation 5-14) would result in a kernel PCA model identical to standard PCA if the original kernel matrix is used. Therefore constructing a low rank approximation of $\mathbf{K}$ with the linear kernel function and comparing the resulting performance to a standard PCA model will give an indication of the performance lost from using low rank approximations.

$$[\mathbf{K}_{linear}]_{ij} = \mathrm{k}_{linear}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \mathbf{x}_j^{\mathrm{T}} \qquad [5\text{-}14]$$

As discussed in section 3.4.1, the kernel width, b, has a large impact on model performance, but it's optimal value is application specific. In the reconstruction algorithm presented in Table 5.8, kernel PCA

is used to calculate the reconstruction error of an observation as the difference between the variance on all components ($\mathbf{A_P}$) and the variance on v retained components ($\mathbf{A_V}$), using equation 3-15. The optimal kernel width is therefore selected as the width that maximizes the variance in the centroid dataset, $\mathbf{L}$, on the v retained components. This is stated formally with equation 5-15:

$$b_{optimal} = \underset{b}{\operatorname{argmax}}\left(\sum_1^v \eta_j\right) \qquad [\text{5-15}]$$

η_j is the fraction of total variance on component j, calculated with equation 3-8.

Table 5.7 gives the model derivation algorithm for kernel PCA. The algorithm is for the target dataset, $\mathbf{X}_t \in \Re^{n_t \times m}$. Step 3 assumes that preceding observations are available for each observation in $\mathbf{X}_t$. Model parameters used when applying the kernel PCA algorithm are highlighted.

Table 5.7: Model derivation algorithm for kernel PCA

Step description		Outputs	Equations
1.	Obtain standardization constants from $\mathbf{X}_t$.	$\boldsymbol{\mu}_t \in \Re^m$ $\boldsymbol{\sigma}_t \in \Re^m$	5-3; 5-4
2.	Standardize $\mathbf{X}_t$ with $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$.	$\mathbf{Z}_t \in \Re^{n_t \times m}$	5-5
3.	Lag the standardized dataset $\mathbf{Z}_t$.	$\mathbf{Z}_t^L \in \Re^{n_t \times m(l+1)}$	3-10
4.	Find the centroid dataset, $\mathbf{L}$.	$\mathbf{L} \in \Re^{k \times m(l+1)}$	5-12*
5.	Find the kernel width that maximizes the significance of v retained components.	$b_{optimal} \in \Re_{>0}$	3-8 5-15
6.	Obtain the kernel matrix, $\mathbf{K}$.	$\mathbf{K} \in \Re^{k \times k}$	3-14 (for Gaussian) 5-14* (for linear)
7.	Centre the kernel matrix.	$\mathbf{K}_c \in \Re^{k \times k}$	3-17
8.	Calculate the eigenvectors- and values of the centred matrix, $\mathbf{K}_c$.	$\mathbf{A} \in \Re^{k \times k}$ $\boldsymbol{\lambda} \in \Re^k$	3-13
9.	Rescale the eigenvectors in $\mathbf{A}$.	$\mathbf{A}' \in \Re^{k \times k}$	5-13*
10.	Construct the matrix of v significant linear component combinations.	$\mathbf{A_V} \in \Re^{k \times v}$	Selected from $\mathbf{A}'$
11.	Construct the matrix of nonzero linear component combinations.	$\mathbf{A_P} \in \Re^{k \times p}$	3-16

Table 5.8 presents the kernel PCA application algorithm, applied to an observation $\mathbf{x_i} \in \Re^m$. Note that step 2 assumes that observations preceding $\mathbf{x}_i$ are known and standardized as in step 1.

Table 5.8: Kernel PCA application algorithm

Step description		Outputs	Equations
1.	Standardize $\mathbf{x}_i$, obtaining $\mathbf{z}_i$.	$\mathbf{z}_i \in \Re^m$	5-5
2.	Lag $\mathbf{z}_i$ with l observations.	$\mathbf{z}_i^L \in \Re^{m(l+1)}$	3-10
3.	Calculate the kernel product of $\mathbf{z}_i$ and $\mathbf{L}$.	$\mathbf{k}_i \in \Re^k$	3-14 (for Gaussian) 5-14* (for linear)
4.	Centre the kernel product.	$\mathbf{k}_{i,\,c} \in \Re^k$	3-18
5.	Estimate the reconstruction error	$\varepsilon_i^R \in \Re$	3-15

Step 4 in Table 5.7 is implemented numerically using MATLAB's built-in *kmeans* function. *kmeans* specifically uses the k-means++ algorithm, and 1000 replicates were used to increase the probability the $\mathbf{L}$ contains the optimal centroids of $\mathbf{Z}_t^L$. Step 5 is implemented using MATLAB's built-in *fmincon* function. Step 8 is executed using the same *eig* function that was used for PCA's model derivation algorithm in Table 5.4. The MATLAB functions used to implement the algorithms presented in this section are given in section E.6 in Appendix E. This includes the algorithm for obtaining the optimal kernel width, $b_{optimal}$.

5.7 Auto-encoder

This section presents the algorithms used for deriving and applying an AE as FPR model. The AE design- and model parameters are given in Table 5.9. This is followed by algorithms for obtaining model parameters (Table 5.10) and applying the derived model to new observations (Table 5.11). Finally, the numerical implementation of the presented AE algorithms in MATLAB is discussed.

Table 5.9: AE design- and model parameters

Design parameter	Symbol	Investigated values
AE architecture	$f(\mathbf{x}_i, \boldsymbol{\theta})$	See Figure 5.8
Bottleneck size	$N_{hidden} \in \mathbb{N}$	$1, 2, 3$
Lag dimension	$l \in \mathbb{N}_0$	$0, 1, 2$
Reconstruction input corruption	σ_C^2	0.10

Model parameters:			
Target data means	$\boldsymbol{\mu}_t \in \Re^m$	Target data standard deviations	$\boldsymbol{\sigma}_t \in \Re^m$
Network parameters	$\boldsymbol{\theta} \in \Re^{N_{\boldsymbol{\theta}}}$		

The basic network architecture used to develop AEs is given in Figure 5.8, and represents the AE model function $f(\mathbf{x}_i, \boldsymbol{\theta})$. Note that the encoding- and decoding layers have twice as many nodes as the input- or output layers. Therefore the number of nodes in the input-, output-, decoding- and encoding layers are set by the number of variables in the modelled data, m, and the lag dimension, l. The only remaining design parameter is the bottleneck layer size.

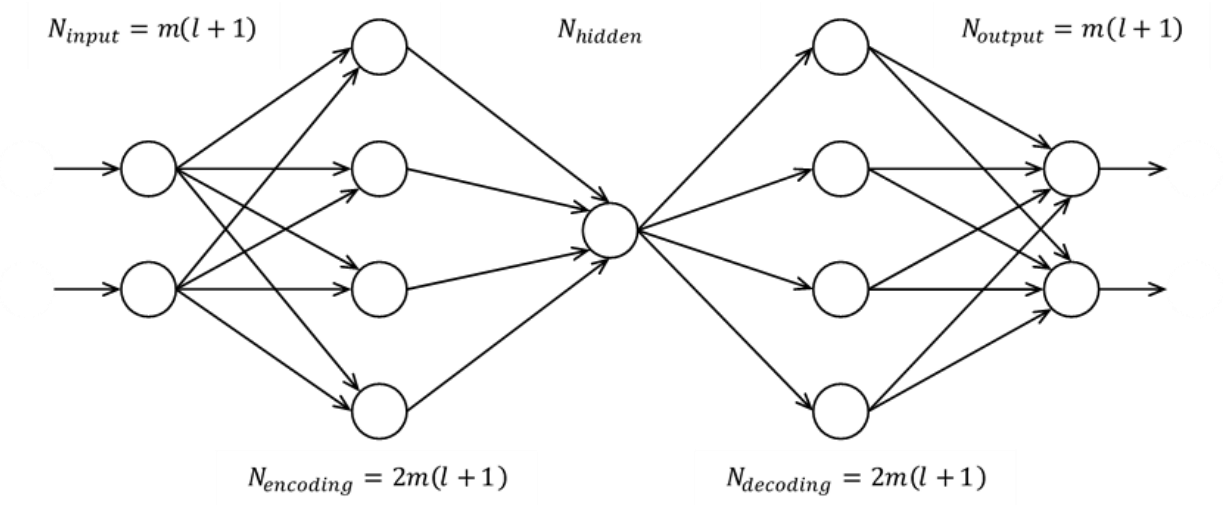

Figure 5.8: Illustration of a lagged AE network architecture. A lagged input, with $m(l+1)$ variables, is projected to a high dimensional encoding layer. The hidden layer extracts representative features from this encoding layer, yielding the nonlinear AE subspace. The decoding- and output layers are used to reconstruct the lagged input from the subspace.

The total number of network parameters, $N_{\boldsymbol{\theta}}$, for the above architecture is calculated with equation 5-16:

$$N_{\boldsymbol{\theta}} = 4\big(m(l+1)\big)^2 + 4N_{hidden}m(l+1) + 4m(l+1) + N_{hidden} \qquad [5\text{-}16]$$

As with kernel PCA, the literature review of AEs (section 3.5) provided high-level descriptions and omitted equations that were unnecessary for understanding AEs, but are applied in the following algorithms to improve the derived model performance. One such equation is target data input corruption (Hu et al., 2020), presented in equation 5-17. Inputs are corrupted with normally distributed noise, and the AE is trained to reconstruct the original, uncorrupted input. This improves model generalizability. The variance of this noise is an additional AE-design parameter, and is included in Table 5.9.

$$\big[\hat{\mathbf{Z}}_t^L\big]_{ij} = [\mathbf{Z}_t^L]_{ij} + N(0, \sigma_C^2) \qquad [5\text{-}17]$$

Regularization is used to modify the error function used when iteratively optimizing AE parameter values (Ng, 2005). L_2-regularization is presented in equation 5-18. Using L_2-regularization, over-fitted parameter values increase the error function. λ is a constant controlling the degree of regularization, with larger values of λ resulting in more regularized model parameters, $\boldsymbol{\theta}$.

$$\mathcal{E}(\mathbf{X}_t, \boldsymbol{\theta})_{L_2} = \mathcal{E}(\mathbf{X}_t, \boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 \qquad [5\text{-}18]$$

Table 5.10 presents the AE model derivation algorithm. Step 3 assumes that preceding observations are available and standardized with which to augment the target data. The final output, $\boldsymbol{\theta} \in \Re^{N_{\boldsymbol{\theta}}}$, is the collection of all parameters in the AE model.

Table 5.10: Model derivation algorithm for AE

	Step description	Outputs	Equations
1.	Obtain standardization constants from $\mathbf{X}_t$.	$\boldsymbol{\mu}_t \in \Re^m$ $\boldsymbol{\sigma}_t \in \Re^m$	5-3; 5-4
2.	Standardize $\mathbf{X}_t$ with $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$.	$\mathbf{Z}_t \in \Re^{n_t \times m}$	5-5
3.	Lag the standardized dataset $\mathbf{Z}_t$.	$\mathbf{Z}_t^L \in \Re^{n_t \times m(l+1)}$	3-10
4.	Construct corrupted $\mathbf{Z}_t$	$\hat{\mathbf{Z}}_t^L \in \Re^{n_t \times m(l+1)}$	5-17*
5.	Optimize model parameters to reconstruct $\mathbf{Z}_t^L$ from $\hat{\mathbf{Z}}_t^L$.	$\boldsymbol{\theta} \in \Re^{N_{\boldsymbol{\theta}}}$	3-21, 5-18

Table 5.11 presents the AE model application algorithm, where an observation $\mathbf{x}_i \in \Re^m$ is reconstructed. Note that the equation applied in step 3 is the optimized AE with the structure given in Figure 5.8 and parameters obtained from the algorithm in Table 5.10.

Table 5.11: AE application algorithm

	Step description	Outputs	Equations
1.	Standardize $\mathbf{x}_i$, obtaining $\mathbf{z}_i$.	$\mathbf{z}_i \in \Re^m$	5-5
2.	Lag $\mathbf{z}_i$ with l observations.	$\mathbf{z}_i^L \in \Re^{m(l+1)}$	3-10
3.	Reconstruct $\mathbf{z}_i^L$.	$\hat{\mathbf{z}}_i^L \in \Re^{m(l+1)}$	$f_{AE}(\mathbf{x}_i, \boldsymbol{\theta})$
4.	Calculate the reconstruction error.	$\varepsilon_i^R \in \Re$	3-3

The model parameter derivation- and reconstruction error algorithms in Table 5.10 and Table 5.11 are implemented numerically in MATLAB. The network training step (step number 4 in Table 5.10) requires additional training parameters to be defined. The first of these is the regularization constant, which has already been introduced in equation 5-18, and is set at $1 \cdot 10^{-5}$ in this investigation. Gradient descent with momentum (equation 3-21) is used as training function, with a learn rate constant equal to 0.01. The loss function was defined as the mean squared error between original (before corruption with equation 5-17) and reconstructed observations. The MATLAB functions used to implement the algorithms in this section are presented in section E.7 in Appendix E.

5.8 Convolutional auto-encoder

This section presents the algorithms used for developing- and applying one-dimensional CAE for FPR in this project. Table 5.12 presents the model- and design parameters for the CAE FPR models investigated. Table 5.13 and Table 5.14 show the algorithms for deriving- and applying model parameters. Table 5.15 and Figure 5.9 provides greater detail on the CAE network architectures developed for this project. This section concludes by discussing how the presented CAE algorithms in Table 5.13 and Table 5.14 are implemented in MATLAB.

Table 5.12: CAE design- and model parameters

Design parameter		**Symbol**	**Investigated values**
CAE network architecture		$f(\mathbf{x}_i, \boldsymbol{\theta})$	See Figure 5.9 and Table 5.15
Reconstruction input corruption		σ_C^2	0.10
Model parameters:			
Target data means	$\boldsymbol{\mu}_t \in \Re^m$	Target data standard deviations	$\boldsymbol{\sigma}_t \in \Re^m$
Network parameters	$\boldsymbol{\theta} \in \Re^{N_{\boldsymbol{\theta}}}$		

Table 5.13 presents the CAE model derivation algorithm. Note that the lagged dataset, $\mathbf{Z}_t^L \in \Re^{m\times(l+1)\times n_t}$, has a different dimension from the lagged datasets for PCA-, kernel PCA- and AE models. Lagged observations are not flattened when training the CAE to preserve the temporal structure of the target dataset, $\mathbf{X}_t$. Network inputs are corrupted during training using equation 5-17, improving generalizability.

Table 5.13: Model derivation algorithm for CAE

	Step description	**Outputs**	**Equations**
1.	Obtain standardization constants from $\mathbf{X}_t$.	$\boldsymbol{\mu}_t \in \Re^m$ $\boldsymbol{\sigma}_t \in \Re^m$	5-3; 5-4
2.	Standardize $\mathbf{X}_t$ with $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$.	$\mathbf{Z}_t \in \Re^{n_t\times m}$	5-5
3.	Lag the standardized dataset $\mathbf{Z}_t$.	$\mathbf{Z}_t^L \in \Re^{(l+1)\times m\times n_t}$	3-10
4.	Construct corrupted $\mathbf{Z}_t$	$\hat{\mathbf{Z}}_t^L \in \Re^{(l+1)\times m\times n_t}$	5-17*
5.	Optimize model parameters to reconstruct $\mathbf{Z}_t^L$ from $\hat{\mathbf{Z}}_t^L$.	$\boldsymbol{\theta} \in \Re^{N_{\boldsymbol{\theta}}}$	3-21

Table 5.14 presents the CAE model application algorithm. Lagged observations are not flattened into single row vectors; this preserves the temporal structure in the MTS segment. The equation applied in step four is the optimized CAE with the network structure presented in Figure 5.9 with network parameters obtained in step 5 in Table 5.13.

Table 5.14: CAE application algorithm

	Step description	Outputs	Equations
1.	Standardize $\mathbf{x}_i$, obtaining $\mathbf{z}_i$.	$\mathbf{z}_i \in \Re^m$	5-5
2.	Lag $\mathbf{z}_i$ with l observations.	$\mathbf{z}_i^L \in \Re^{(l+1)\times m}$	3-10
3.	Reconstruct $\mathbf{z}_i^L$.	$\hat{\mathbf{z}}_i^L \in \Re^{(l+1)\times m}$	$f_{CAE}(\mathbf{x}_i, \boldsymbol{\theta})$
4.	Calculate the reconstruction error.	$\varepsilon_i^R \in \Re$	3-3

Two convolutional auto-encoder architectures were investigated in this project. These architectures are presented in Figure 5.9, and they are distinguished from the lagged auto-encoder architecture given in Figure 5.8 by first extracting features across the time dimension and then extracting features across the measured variables.

The short convolutional auto-encoder architecture contains one fewer hidden layer used to convolve across the time dimension than the long convolutional auto-encoder. It is expected that the longer architecture would extract deeper features across the time dimension, due to its extra hidden layer and the fact that it processes six lagged values in the input a.o.t. the four lagged for the short architecture.

Note that Figure 5.9 does not illustrate the full dimensionality of the weights- and biases within the two developed architectures; it only shows how convolution operations were applied. Table 5.15 provides a more in depth description of the dimension of each hidden layer and the filters applied to them.

The convolutional auto-encoder architectures presented in Table 5.15 and Figure 5.9, as well as the algorithms in Table 5.13 and Table 5.14 used to apply them for FPR were implemented numerically in MATLAB. The stochastic gradient descent with momentum algorithm (given in equation 3-21) is employed to optimize the network parameters in Table 5.15. As with the auto-encoder modelling approach given in the previous section, L_2-regularization was employed, with $\lambda = 0.01$. The loss function was defined as the mean squared error between the original input and its reconstruction. The MATLAB functions used to implement the algorithms in this section are presented in section E.8 in Appendix E.

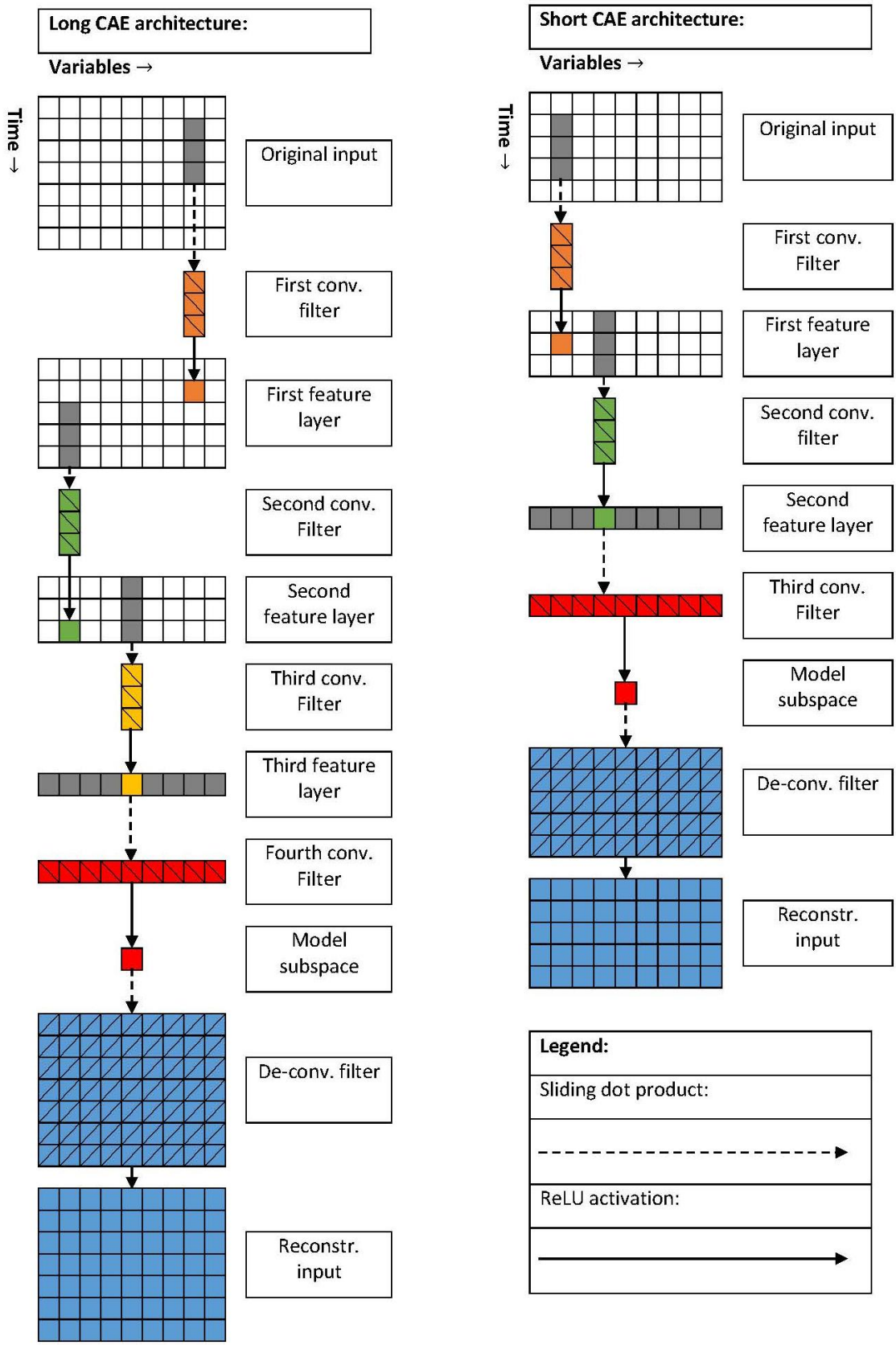

Figure 5.9: Illustration of the long- (left) and short (right) CAE architectures evaluated in this project. Convolutions that are applied vertically convolve a feature in the time dimension. Horizontal convolutions convolve across the variables in a feature.

Table 5.15: CAE network architectures

Long CAE network architecture parameters:					
	Layer type	Output size	Filter shape	No. of filters	Learnable parameters
1.	Input	$7 \times 9 \times 1$	-	-	-
2.	Convolution + ReLU	$5 \times 9 \times 8$	3×1	8	$Weights$: $3 \times 1 \times 1 \times 8$ $Biases$: 8
3.	Convolution + ReLU	$3 \times 9 \times 8$	3×1	8	$Weights$: $3 \times 1 \times 8 \times 8$ $Biases$: 8
4.	Convolution + ReLU	$1 \times 9 \times 8$	3×1	8	$Weights$: $3 \times 1 \times 8 \times 8$ $Biases$: 8
5.	Convolution + ReLU	$1 \times 1 \times 4$	1×9	4	$Weights$: $1 \times 9 \times 8 \times 4$ $Biases$: 4
5.	Deconvolution	$7 \times 9 \times 1$	7×9	1	$Weights$: $7 \times 9 \times 4 \times 1$ $Biases$: 1
6.	Output	$7 \times 9 \times 1$	-	-	-
Total parameters in long CAE structure: **977**					
Short CAE network architecture parameters:					
	Layer type	Output size	Filter shape	No. of filters	Learnable parameters
1.	Input	$5 \times 9 \times 1$	-	-	-
2.	Convolution + ReLU	$3 \times 9 \times 8$	3×1	8	$Weights$: $3 \times 1 \times 1 \times 8$ $Biases$: 8
3.	Convolution + ReLU	$1 \times 9 \times 8$	3×1	8	$Weights$: $3 \times 1 \times 8 \times 8$ $Biases$: 8
4.	Convolution + ReLU	$1 \times 1 \times 4$	1×9	4	$Weights$: $1 \times 9 \times 8 \times 4$ $Biases$: 4
5.	Deconvolution	$5 \times 9 \times 1$	5×9	1	$Weights$: $7 \times 9 \times 4 \times 1$ $Biases$: 1
6.	Output	$5 \times 9 \times 1$	-	-	-
Total parameters in short CAE structure: **707**					

6 RESULTS AND DISCUSSION

This chapter addresses the third objective identified for this project: the monitoring performance of the FPR models developed in the previous chapter are evaluated and compared. All of the developed models are reconstruction-based, and therefore the $AUC_{ROC,\psi_{min}}$-value describes the quality of discriminant statistics generated by the presented FPR models and allows FPR performance to be presented concisely in a single metric. Figure 6.1 and Table 6.1 shows the optimal $AUC_{ROC,\psi_{min}}$-value achieved by each model type on the testing data, with- and without first applying feature engineering to the data.

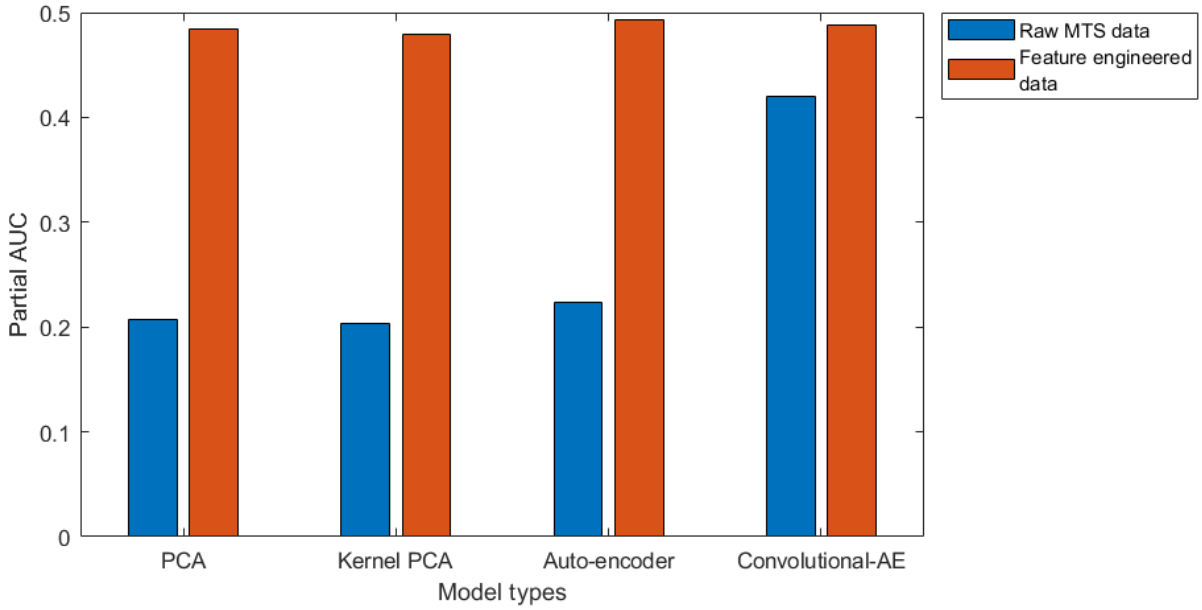

Figure 6.1: Illustration of the impact of feature engineering on model performance. The optimal partial AUC value obtained on the testing dataset is plotted for each model type, with- and without feature engineering techniques being applied. Note that the optimal $AUC_{ROC,\psi_{min}}$ on the raw MTS data for all investigated models excluding the convolutional auto-encoder is less than half the optimal $AUC_{ROC,\psi_{min}}$ obtained from the engineered dataset.

Table 6.1: Maximum $AUC_{ROC,\psi_{min}}$-values achieved for each model type, applied to raw MTS data and feature engineered data.

Model type	$AUC_{ROC,\psi_{min}}$, **raw MTS data**	$AUC_{ROC,\psi_{min}}$, **engineered data**
PCA	0.207	0.485
Kernel PCA	0.204	0.479
Auto-encoder	0.224	0.493
Convolutional auto-encoder	0.419	0.488

Salfner et al. (2010) noted that feature engineering generally has a greater impact on FPR performance than model configuration (w.r.t. model type- and parameters). The trends observed in Figure 6.1 confirm this; the optimal $AUC_{ROC,\psi_{min}}$ more than doubles for the PCA-, kernel PCA- and AE-models developed if

feature engineering has been applied. An obvious exception is the convolutional AE-model, which has by far the best $AUC_{ROC,\psi_{min}}$-value when applied to the raw MTS data. Therefore, this chapter will evaluate each FPR model's performance on the feature engineered data, with the exception of the convolutional AE-model, which will be evaluated on both the feature engineered- and raw MTS data.

The performance of the PCA models developed in this project is discussed in section 6.1. This section discusses the optimal model configuration of the developed PCA models, applied to both raw- and MTS data. This section shows how the feature engineering techniques employed converted the dynamic- high variance signals generated by the furnace model to low variance-static representation. This section also shows that lagging this static representation is detrimental to FPR performance.

The reviewed literature suggested that Gaussian kernel PCA is a superior monitoring model to standard linear kernel PCA, but Figure 6.1 shows that Gaussian kernel PCA's performance is almost identical to standard linear PCA. Section 6.2 discusses this discrepancy, and shows that the k-means clustering approximation used as part of the kernel PCA algorithm presented in section 5.6 causes a drop in performance, and confirms that Gaussian kernel PCA's FPR performance is improved by nonlinear feature extraction. Section 6.2 also discusses how kernel PCA's model configuration impacts FPR performance.

The auto-encoder FPR model type displays the best performance when applied to the feature engineered data. Section 6.3 expands on the auto-encoder configuration and evaluates how the lag dimension and hidden layer size impacts the FPR performance of an auto-encoder model.

Section 6.4 discusses the convolutional auto-encoder FPR performance, and describes how the convolutional auto-encoder structure selected (as presented in Table 5.15 and Figure 5.9 in section 5.8) impacts FPR performance. This section finds that the one-dimensional convolution operation is better for extracting dynamic characteristics than simply lagging process data.

$AUC_{ROC,\psi_{min}}$-values allow optimal FPR models to be selected comparatively, but an individual model's performance can be better understood in terms of absolute performance metrics. Therefore, section 6.5 expands on the FPR performance observed for the AE-model applied to feature engineered data, as this model achieved the highest $AUC_{ROC,\psi_{min}}$-value, as well as the FPR performance for the CAE-model applied to raw MTS data. This section evaluates how these models perform in terms of missed predictions, detection delays and false alarm rates.

6.1 PCA model evaluation

Table 6.2 provides a summary of the optimal design parameters for recognizing blowback-preceding conditions using PCA. It also presents the $AUC_{ROC,\psi_{min}}$-value when applying each model to the training-validation- and testing data (as partitioned in section 5.2).

Table 6.2: Summary of optimal design parameters for the PCA FPR model

Design parameter	**Investigated levels**	**FE data**
Retained components, v	$0, 1, \dots, 7, 8$	8
Lag dimension, l	$0, 1, 2, \dots, 9, 10$	0
Recognition window length, l_w	$0, 30, \dots, 330, 360$	90
Feature engineered data FPR performance		
Dataset		$AUC_{ROC,\,\psi_{min}}$
Train		0.546
Validation		0.514
Test		0.485

Table 6.2 shows that the $AUC_{ROC,\,\psi_{min}}$-value for each configuration changes from the training- testing and validation data. Figure 6.2 plots the $AUC_{ROC,\,\psi_{min}}$-value for each of these datasets.

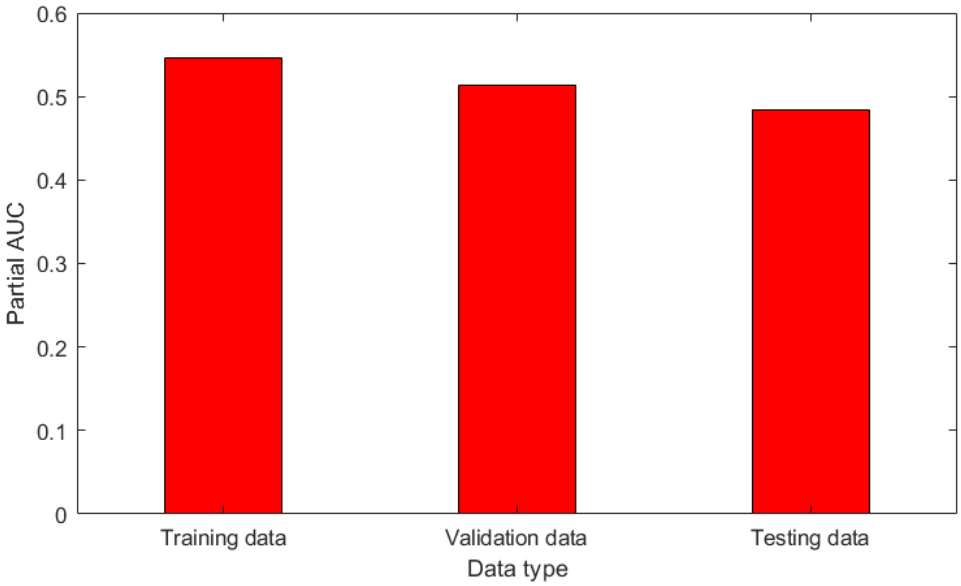

Figure 6.2: Illustration of PCA model generalizability. The $AUC_{ROC,\,\psi_{min}}$-value is plotted for the training-, validation- and testing data.

Figure 6.2 shows that the optimal PCA FPR model generalizes well beyond the training set. However, the $AUC_{ROC,\,\psi_{min}}$-value obtained on the training set (0.546) does diminish to 0.485 on the testing set, confirming that PCA FPR models are not immune to over-fitting, and confirms that the model should be validated and tested separately.

Table 6.2 shows that the optimal PCA model on the feature engineered data has a maximum performance for a recognition window length equal to 90 samples. This corresponds to 15 minutes of observations. Figure 6.3 illustrates how $AUC_{ROC,\,\psi_{min}}$-values vary with recognition window length for the configuration given in Table 6.2.

Figure 6.3 shows that evaluating discriminant values in a moving window, as described in section 3.7.2, has a noticeable impact on FPR performance. The $AUC_{ROC,\psi_{min}}$-value of the PCA model increases from 0.391 when evaluating single discriminant values to 0.485 at $l_w = 90$.

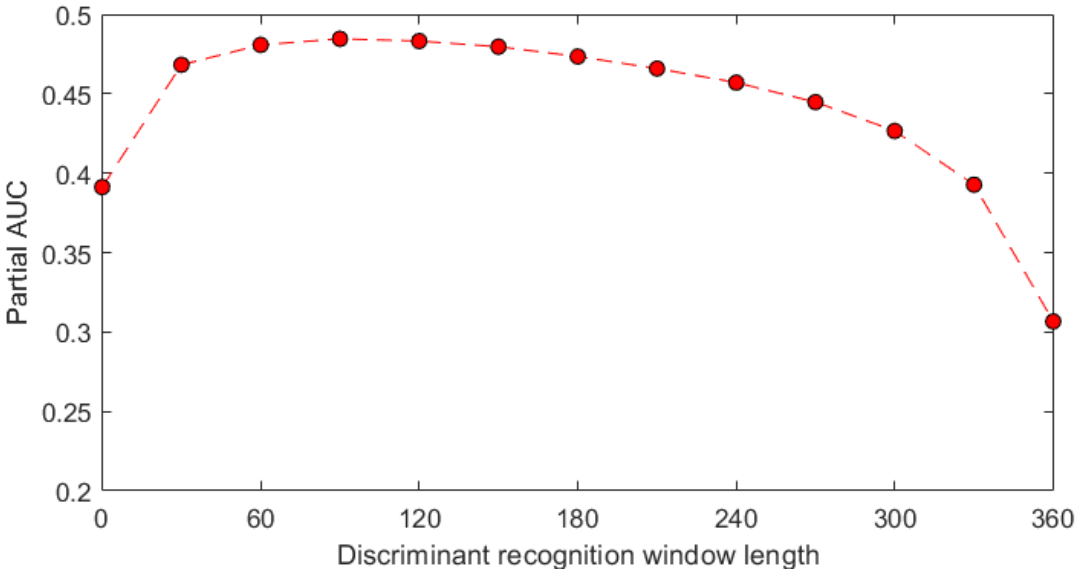

Figure 6.3: Illustration of $AUC_{ROC,\psi_{min}}$-values obtained for the optimal PCA configuration in Table 6.2 with varying recognition window length. The red markers show the $AUC_{ROC,\psi_{min}}$-value obtained at each recognition window length investigated.

The number of retained components for the optimal PCA model configuration given in Table 6.2 is at the maximum possible number, as retaining one more component would cause reconstructed observations to be equal to the original (and no reconstruction error can be calculated). This suggests that increasing the number of retained components increases the ratio of reconstruction error between faulty- and fault free observations. This is illustrated in Figure 6.4.

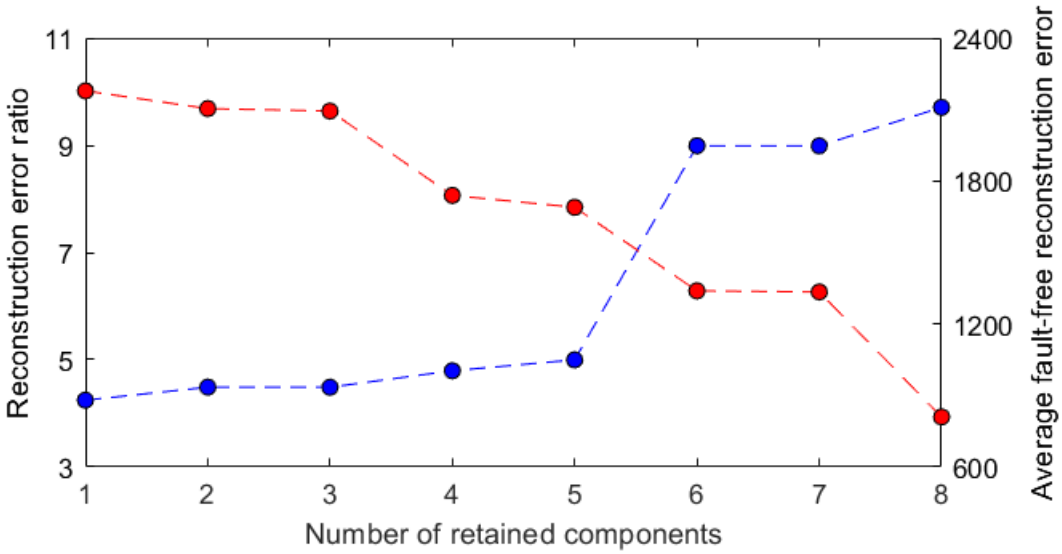

Figure 6.4: Illustration of the effect of retained components on reconstruction error- and reconstruction error ratio. The red markers show the decreasing average reconstruction error of fault-free observations

in the testing dataset as the number of retained components increase. Blue markers show the increasing reconstruction error ratio between fault-free and faulty observations.

Figure 6.4 shows that the ratio between the average reconstruction error for fault-free observations- and faulty observations increase as more components are retained. Reconstruction-based FPR models rely on larger reconstruction errors for fault-free than faulty observations, hence retaining more principal components would therefore increase FPR performance. The ROC-curve for the optimal PCA model, with the number of retained components varied, is presented in Figure 6.5 to confirm that retaining more components does indeed improve sensitivity.

Figure 6.5 confirms that increasing the number of retained components improves sensitivity. However, the minimum sensitivity for 95% precision can be achieved for each of the evaluated number of retained components. Figure 6.5 omits ROC curves for most of the investigated retained components for readability. Figure 6.6 shows the $AUC_{ROC,\,\psi_{min}}$-values obtained for each number of retained principal components for the optimal PCA model configuration, confirming that increasing the number of retained principal components increases the $AUC_{ROC,\,\psi_{min}}$-value for the presented PCA model.

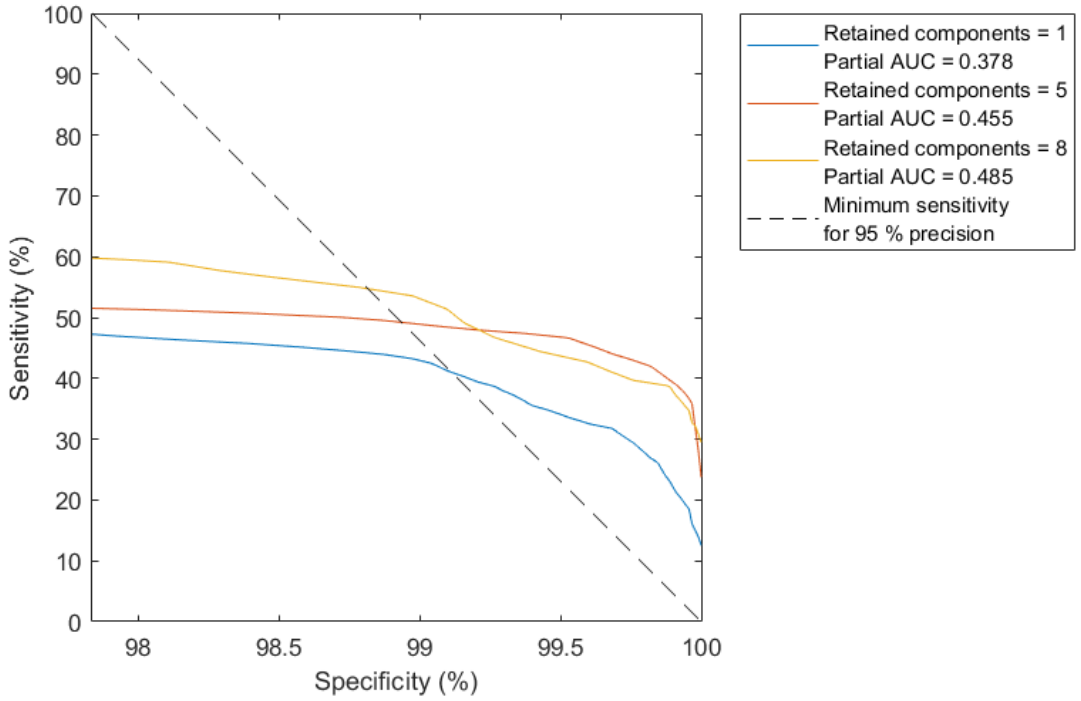

Figure 6.5: ROC curve of the PCA model applied to feature engineered data with the optimal design parameter configuration as presented in Table 6.2, with the number of retained principal components varied.

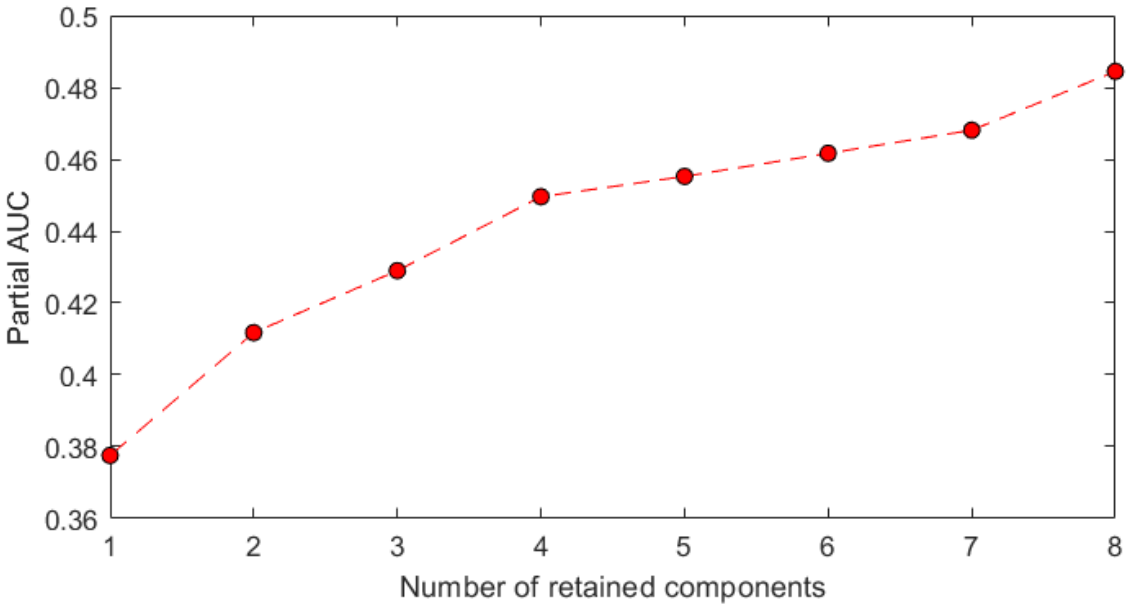

Figure 6.6: $AUC_{ROC,\psi_{min}}$-values obtained for varying number of retained components retained in the optimal PCA model configuration. Red markers show $AUC_{ROC,\psi_{min}}$-values obtained at each number of retained components.

Table 6.2 shows that the $AUC_{ROC,\psi_{min}}$-value of the PCA model is at the optimum when no lag is included. This is unexpected, because adding lagged variables allows PCA to extract auto-correlations. However, evaluating the feature engineering techniques employed may shed light on this discrepancy. Feature engineering techniques were selected to replace high variance signals with static representations, replacing the dynamic characteristics.

The partial auto-correlation function is used to confirm that feature engineering has replaced dynamic behaviours with static representations; it shows how a variable is auto-correlated with its lagged values, while removing correlations between lagged values. A partial auto-correlation plot is generated for both the freeboard furnace pressure data generated by the furnace model and the engineered pressure signal. The resulting graph is given in Figure 6.7.

Figure 6.7 shows that the raw pressure signal is correlated with itself, as its partial auto-correlation function is greater than zero various lags. The partial auto-correlation plot shows that the engineered signal is as strongly correlated with the value immediately preceding it as it is with itself (both have function values equal to 1), and that it is uncorrelated with values at larger lags (indicated by function values lower than zero). This shows that the engineered signal is a static representation of the original dynamic signal.

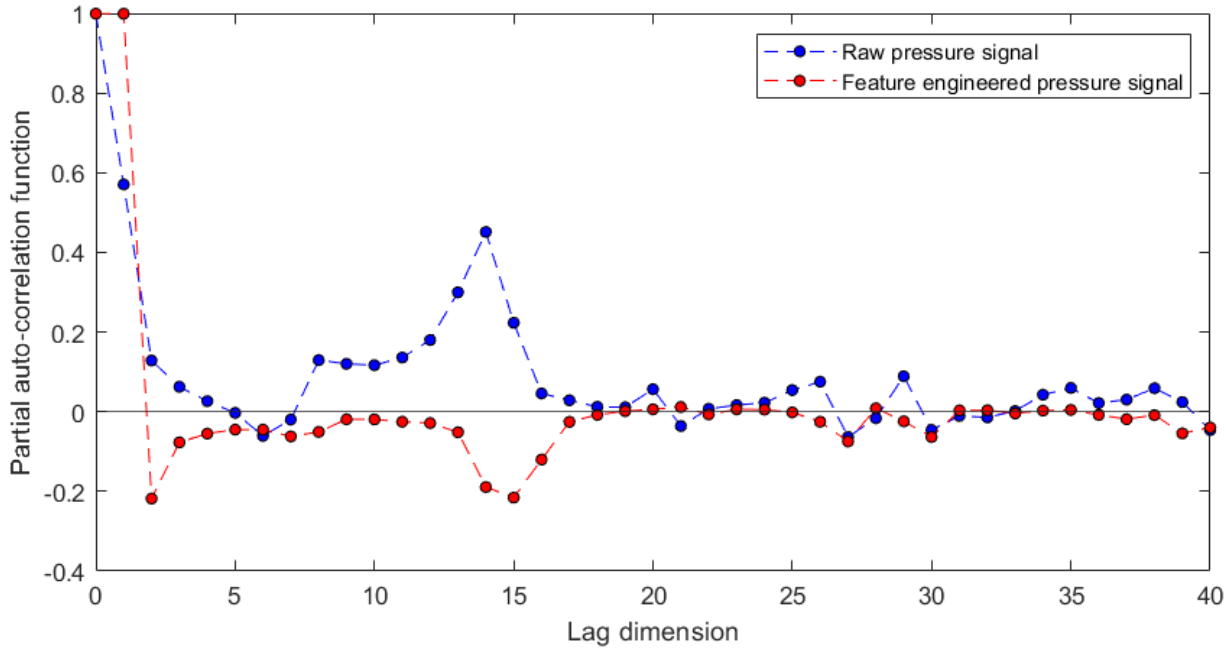

Figure 6.7: Partial auto-correlation for the raw freeboard pressure data (blue markers) and for the feature engineered data (red markers) over 40 lags.

Figure 6.8 presents the ROC-curve for the PCA model with the configuration given in Table 6.2, and with the lag dimension varied. Figure 6.8 shows that lagging the feature engineered data reduces FPR performance, but not by much, as the minimum sensitivity for 95% precision can be achieved comfortably at all investigated lag dimensions. Figure 6.8 omits ROC curves for most of the investigated lag dimensions for readability. Figure 6.9 shows the $AUC_{ROC,\,\psi_{min}}$-values obtained for each lag dimension at the optimal PCA model configuration.

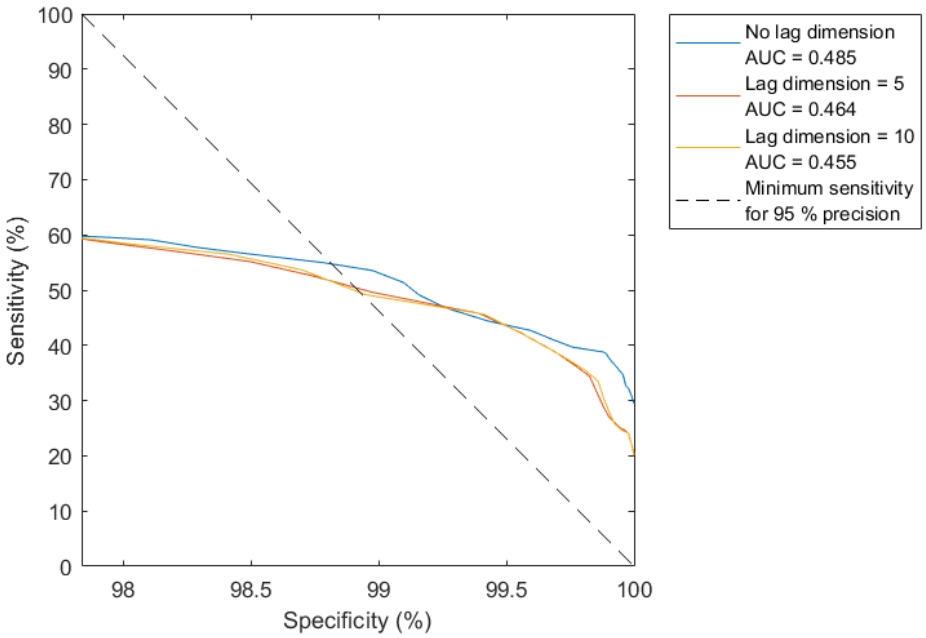

Figure 6.8: ROC curve of the PCA model applied to feature engineered data with the optimal design parameter configuration as presented in Table 6.2, with the lag dimension varied.

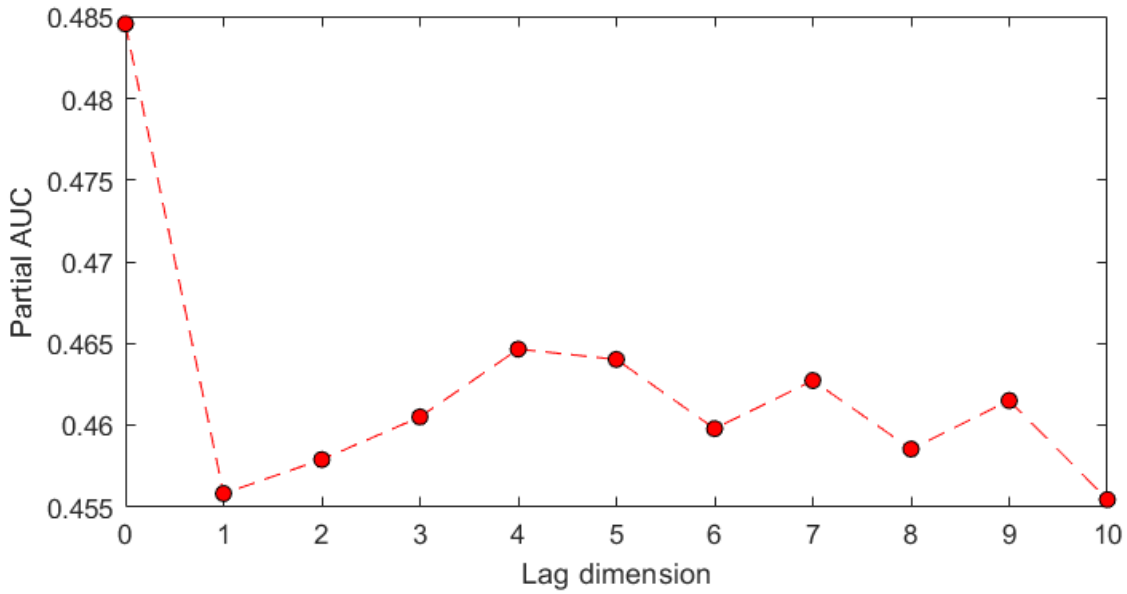

Figure 6.9: $AUC_{ROC,\,\psi_{min}}$-values obtained the optimal PCA model configuration with varying lag dimension. Red markers show $AUC_{ROC,\,\psi_{min}}$-values obtained at each lag dimension.

6.2 Kernel PCA

Table 6.3 summarizes the optimal design parameters for recognizing blowback-preceding conditions using kernel PCA. It also shows the $AUC_{ROC,\,\psi_{min}}$-value obtained from the training- validation- and testing data.

Table 6.3: Summary of optimal design parameters for kernel PCA FPR models

Design parameter	**Investigated levels**	**Optimal parameter value**
Retained components, v	$0, 1, \dots, 9, 10$	6
Lag dimension, l	$0, 1, 2, \dots, 5,\ 6$	0
Recognition window length, l_w	$0, 30, \dots, 330, 360$	60
Feature engineered data FPR performance		
Dataset	$AUC_{ROC,\,\psi_{min}}$	
Train	0.506	
Validation	0.494	
Test	0.479	

Table 6.3 shows that the kernel PCA FPR model is well-generalized when applied to the feature engineered data. However, this performance is poorer than the performance observed for the standard PCA model presented in Table 6.2, and this is illustrated in Figure 6.10. Kernel PCA was expected to outperform standard linear PCA due to its ability to extract nonlinear correlations, but standard linear PCA outperforms it on each data type.

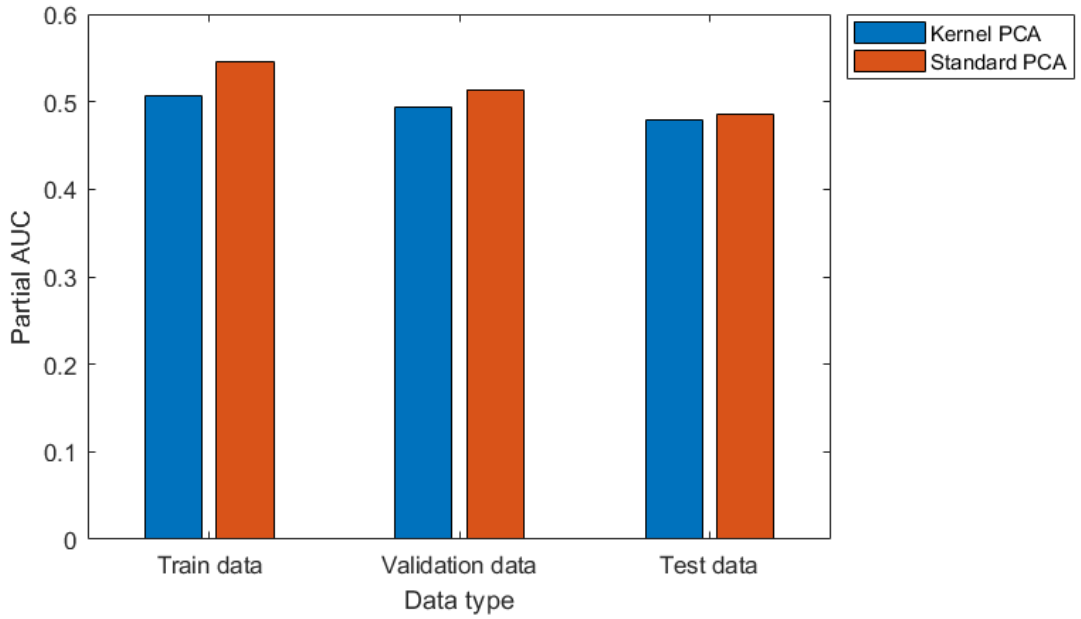

Figure 6.10: Comparison of Gaussian kernel PCA- and standard linear PCA FPR performance on training-, validation- and testing data. Linear PCA demonstrates superior performance to Gaussian kernel PCA on each dataset.

The kernel matrix approximation used in step 4 of the kernel PCA algorithm in Table 5.7 is a likely cause of this discrepancy. Most of the reviewed kernel PCA applications worked on small datasets where approximation is unnecessary; Lee et al. (2004) and Choi and Lee (2004) modelled a simulated wastewater treatment process using 672 samples, Deng and Tian (2013) did the same for the Tennessee Eastman process using 480 samples. The dataset considered here is comparatively large (although still miniscule to industry datasets), with nearly ten thousand samples in the target dataset and more than 240 000 samples in the testing dataset. The kernel matrix had to be approximated with k-means clustering, and approximation would reduce performance (Schölkopf et al., 1998a).

The linear kernel function, introduced in section 5.6, yields a model identical to standard PCA if the full kernel matrix is constructed. The impact of approximation on kernel PCA performance can be illustrated by comparing a standard PCA model with an approximated linear kernel PCA model. Figure 6.11 presents the ROC curve obtained for the optimal Gaussian kernel PCA model, optimal linear PCA model and the linear kernel PCA model evaluated with the optimal design parameters for PCA presented in Table 6.2.

Figure 6.11 shows that the Gaussian kernel PCA model is superior to its linear counterpart. This suggests that the simulated fault patterns contain characteristic nonlinear correlations, and confirms that modelling these nonlinear correlations yields improved FPR performance. Figure 6.11 also illustrates that nonlinear correlations obtained from approximated data are less effective at characterizing fault patterns than linear correlations obtained from the actual data; standard PCA, with $AUC_{ROC,\psi_{min}} = 0.485$, performs better than Gaussian kernel PCA.

The results presented in this section highlights a key weakness of kernel PCA that cannot be noticed in small datasets; it is a data-driven model that struggles with large volumes of data. Kernel PCA models are

too computationally expensive to apply on large datasets and low-rank approximations of large datasets fail to capture all significant characteristics, causing drops in FPR performance. This drawback to kernel PCA is expected to be more pronounced in industrial datasets; the simulated dataset considered here was generated for 12 weeks of SAF operation, while industry datasets are recorded over multiple years.

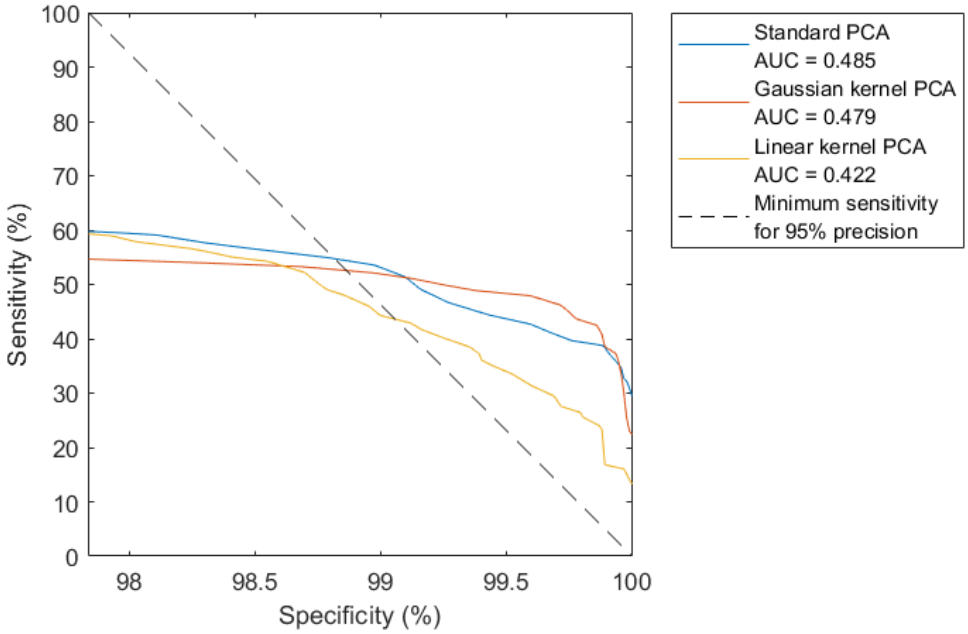

Figure 6.11: ROC curves of standard PCA, Gaussian kernel PCA- and linear kernel PCA FPR models when applied to feature engineered data.

Table 6.3 shows that kernel PCA achieves its optimal performance for a recognition window length equal to 60, corresponding to 10 minutes of discriminant samples. However, evaluating the discriminant values in a window did not yield the same increases in performance as was observed with standard PCA; the $AUC_{ROC,\,\psi_{min}}$-value increased from 0.475 at $l_w = 0$ to 0.479 at $l_w = 60$.

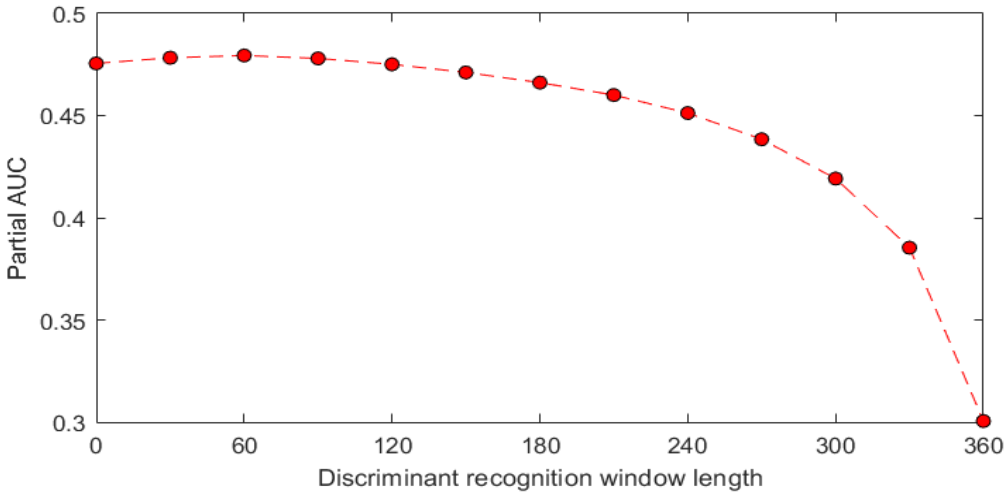

Figure 6.12: Illustration of $AUC_{ROC,\,\psi_{min}}$-values obtained for the optimal kernel PCA configuration in Table 6.3 with varying recognition window length. The red markers show the $AUC_{ROC,\,\psi_{min}}$-value obtained at each recognition window length investigated.

The optimal kernel PCA model configuration given in Table 6.3 has eight retained components. Figure 6.13 illustrates how kernel PCA FPR sensitivity varies for a select number of retained principal components. Figure 6.14 presents $AUC_{ROC,\psi_{min}}$-values for all retained principal component levels investigated for kernel PCA. Both Figure 6.13 and Figure 6.14 show that a clear optimum in the number of retained principal components exist; unlike with the linear PCA model, this optimum is not found at the maximum number of retainable components.

The optimal kernel PCA configuration in Table 6.3 also indicates that lagged variables reduce kernel PCA FPR performance. This again suggests that the feature engineered data do not contain the dynamic characteristics of the raw MTS data, therefore adding lagged values to modelled data does not facilitate better FPR performance. This is illustrated in Figure 6.15 and Figure 6.16; Figure 6.15 presents the ROC-curves for the optimal kernel PCA model configuration, as well as the ROC-curves for varying the lag dimension. Figure 6.15 shows that the kernel PCA model achieves the largest sensitivities when variables are not lagged. Figure 6.16 expands on Figure 6.15 by showing the $AUC_{ROC,\psi_{min}}$-values achieved for all investigated lag dimensions, and confirms that adding lag dimensions lowers sensitivitity.

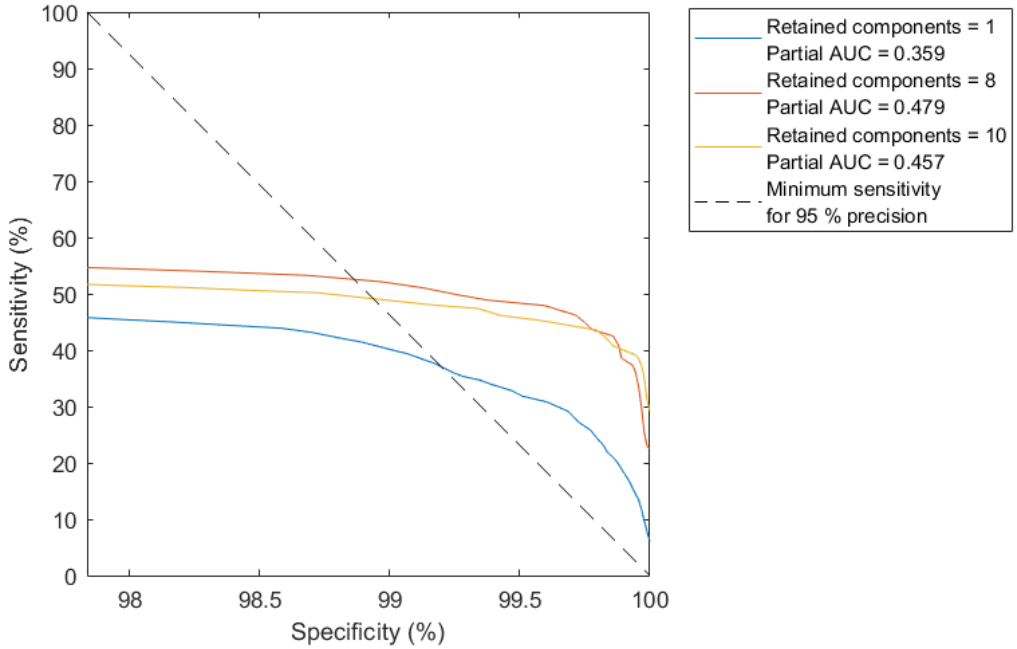

Figure 6.13: ROC curve of the kernel PCA model applied to feature engineered data with the optimal design parameter configuration as presented in Table 6.3, with the number of retained principal components varied.

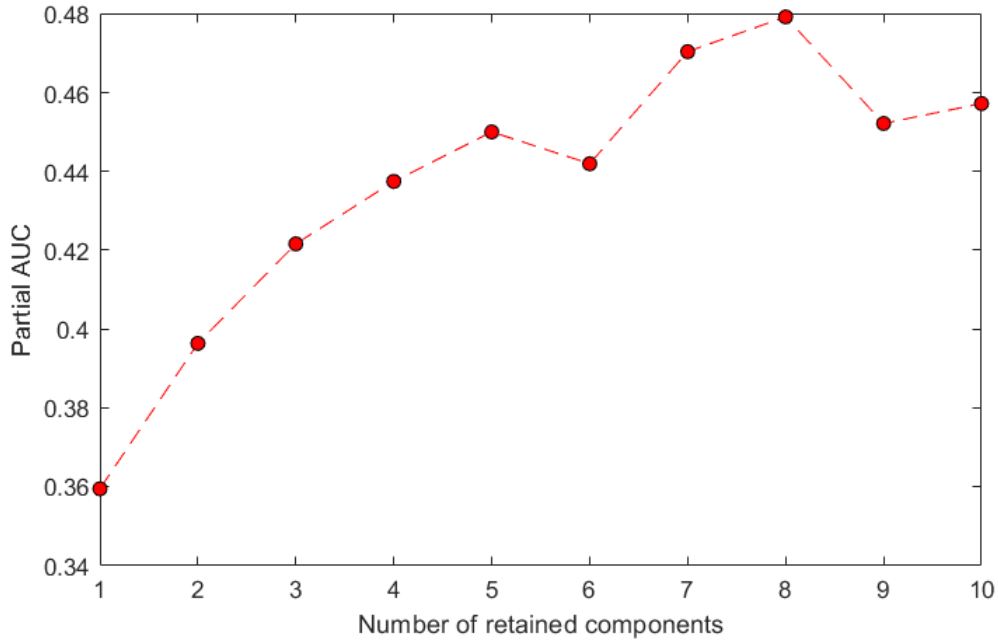

Figure 6.14: $AUC_{ROC,\psi_{min}}$-values obtained for varying number of retained components retained in the optimal kernel PCA model configuration. Red markers show $AUC_{ROC,\psi_{min}}$-values obtained at each number of retained components.

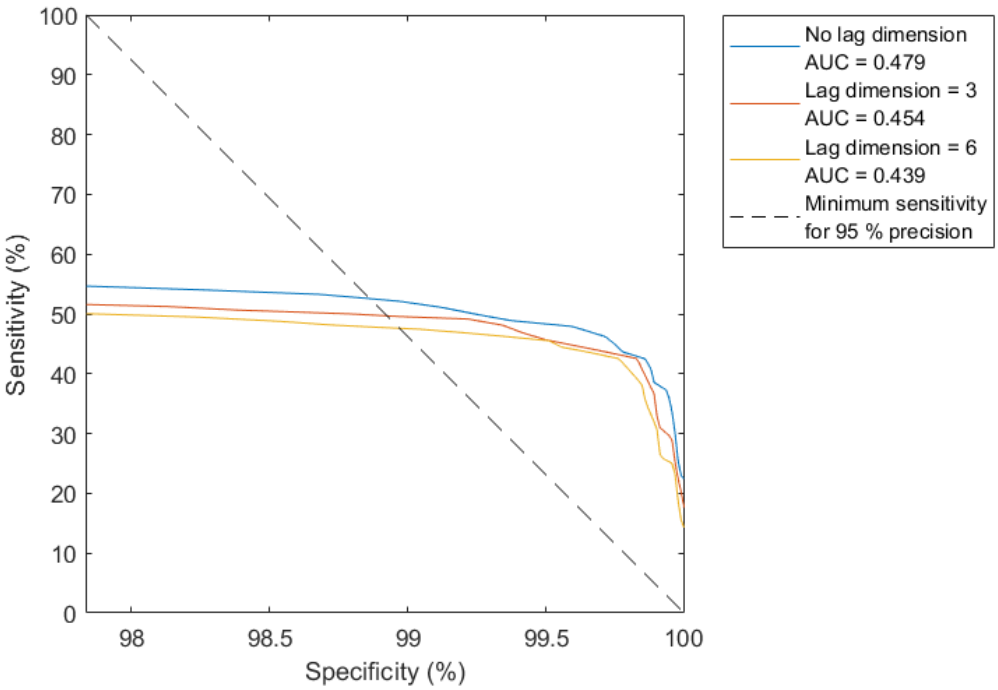

Figure 6.15: ROC curve of the kernel PCA model applied to feature engineered data with the optimal design parameter configuration as presented in Table 6.3, with the lag dimension varied.

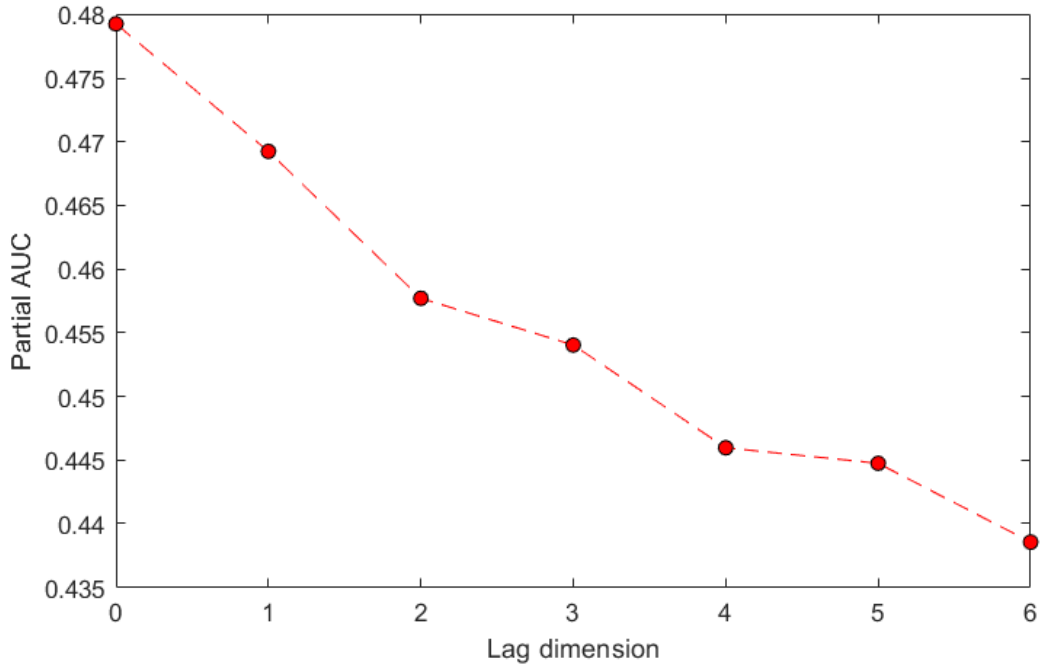

Figure 6.16: $AUC_{ROC,\,\psi_{min}}$-values obtained the optimal kernel PCA model configuration with varying lag dimension. Red markers show $AUC_{ROC,\,\psi_{min}}$-values obtained at each lag dimension.

6.3 Auto-encoder

Table 6.4 summarizes the optimal design parameters for recognizing blowback-preceding conditions using auto-encoders. It also shows the $AUC_{ROC,\,\psi_{min}}$-value obtained from the training- validation- and testing data.

Table 6.4: Summary of optimal design parameters for AE FPR models

Design parameter	**Investigated levels**	**Optimal parameter value**
Hidden layer size, N_{hidden}	$1, 2, 3$	2
Lag dimension, l	$0, 1, 2$	0
Recognition window length, l_w	$0, 30, \ldots, 330, 360$	120
Feature engineered data FPR performance		
Dataset	$AUC_{ROC,\,\psi_{min}}$	
Train	0.535	
Validation	0.492	
Test	0.493	

Table 6.4 shows that the AE model generalizes well beyond the training set, its $AUC_{ROC,\,\psi_{min}}$-value decreases from 0.535 when applied to the training data to 0.493 when applied to the testing data. Figure 6.17 illustrates the AE-model generalizability.

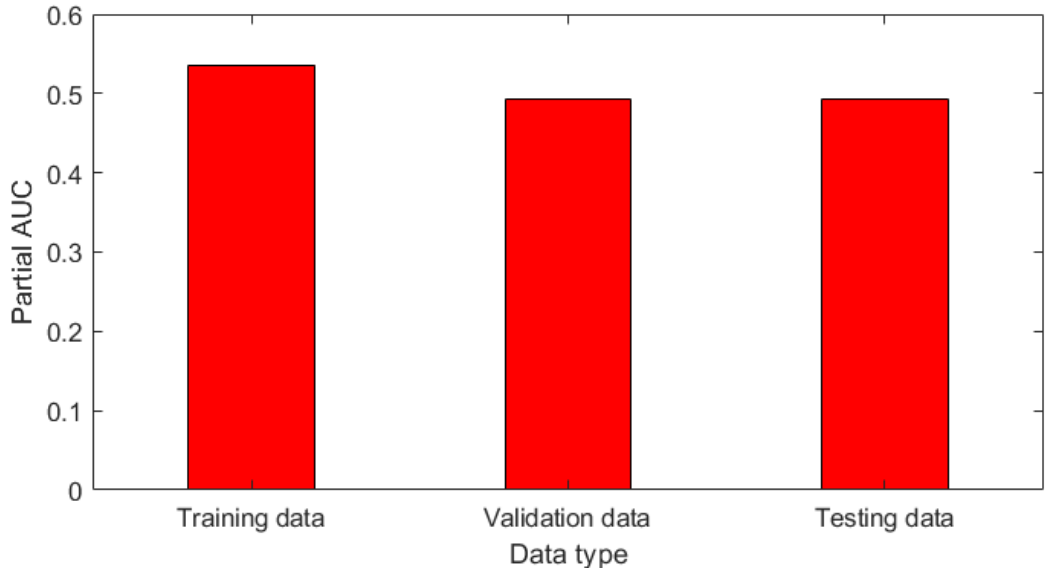

Figure 6.17: Illustration of AE model generalizability. The $AUC_{ROC,\psi_{min}}$-value is plotted for the training-, validation- and testing data.

Table 6.4 also shows that the optimal recognition window length for the optimal AE model is at $l_w = 120$, corresponding to 20 minutes of observations, where an $AUC_{ROC,\psi_{min}}$-value of 0.493 is obtained. Figure 6.18 illustrates how $AUC_{ROC,\psi_{min}}$-values vary over all evaluated recognition window lengths.

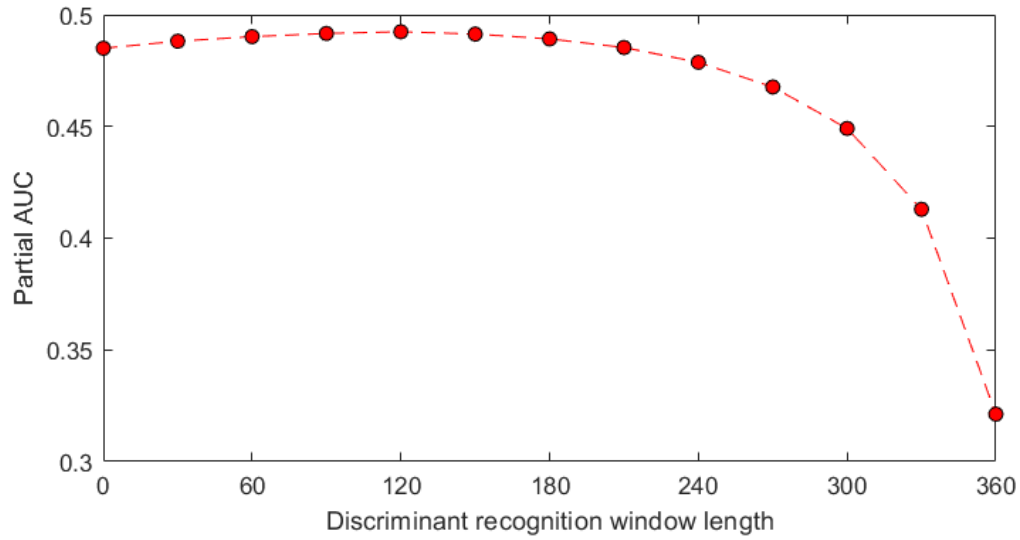

Figure 6.18: Illustration of $AUC_{ROC,\psi_{min}}$-values obtained for the optimal AE configuration in Table 6.4 with varying recognition window length. The red markers show the $AUC_{ROC,\psi_{min}}$-value obtained at each recognition window length investigated.

Figure 6.18 shows that evaluating discriminant values in a window does not have the strong impact on AE FPR performance that is observed for standard PCA. For a standard PCA FPR model, moving window discriminant evaluation increases the $AUC_{ROC,\psi_{min}}$-value from 0.391 to 0.485, by contrast, an AE FPR model only increased from 0.485 to 0.493.

According to Table 6.4, the optimal hidden layer size identified for the AE-FPR model is 2. The literature review on AE-FPR models suggested that this hidden layer size is a crucial design parameter, and the effect of varying this design parameter is illustrated in the ROC curves given in Figure 6.19. Figure 6.19 confirms that the AE model's FPR performance is very sensitive to this hidden layer size; increasing the hidden layer size to 3 results in an AE model being unable to meet the required sensitivity for 95 % precision. This suggests that using too many neurons in the hidden layer causes the AE output to be an equivalent representation of its input, with detrimental effects on FPR performance.

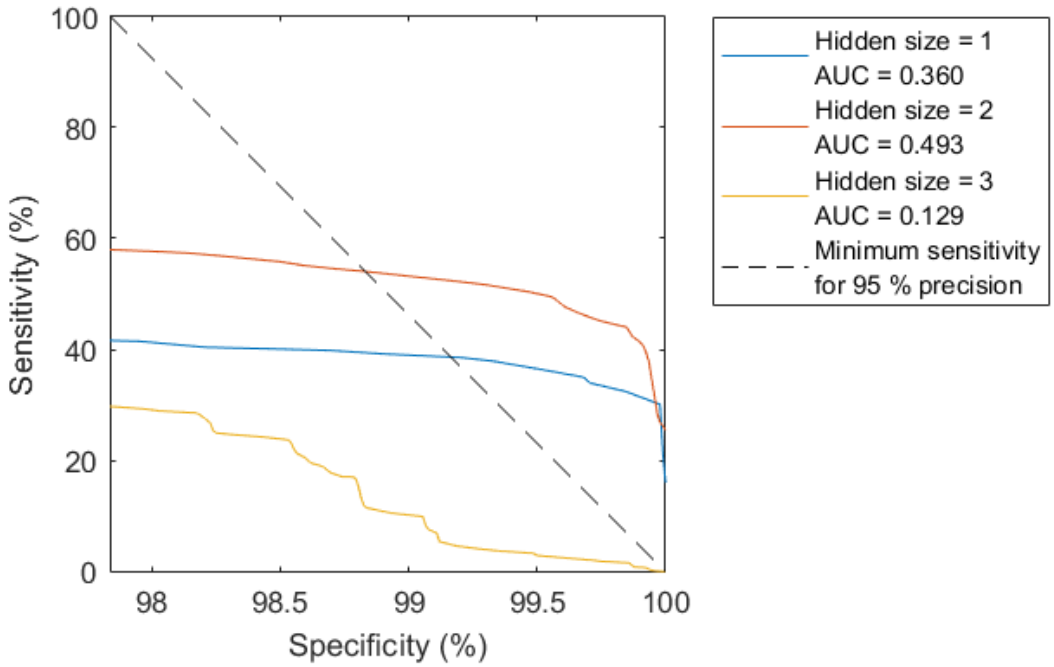

Figure 6.19: ROC curve of the AE model applied to feature engineered data with the optimal design parameter configuration as presented in Table 6.4, with the hidden layer size varied.

The results presented for PCA- and kernel PCA in sections 6.1 and 6.2 both suggested that the feature engineered data is a static representation of the dynamic data, and that adding lagged values would therefore not improve FPR performance. Figure 6.20 presents the ROC-curve for the optimal AE FPR model configuration, with varying lag dimensions. Figure 6.20 reinforces the trends observed in sections 6.1 and 6.2; adding lagged variables to the feature engineered data slightly reduces FPR performance, reflected in a loss in sensitivity as lags are added.

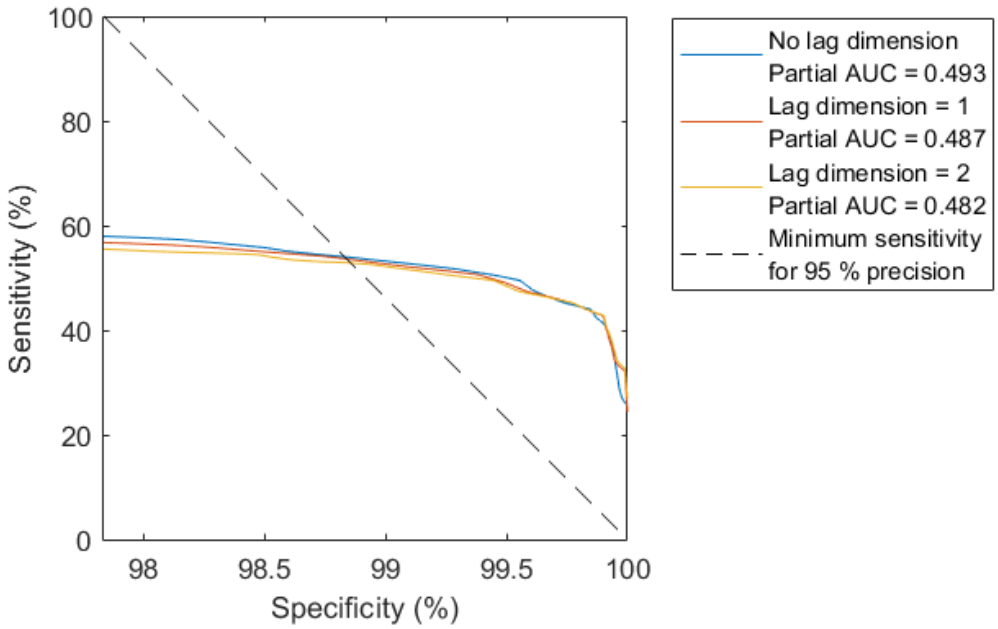

Figure 6.20: ROC curve of the AE model applied to feature engineered data with the optimal design parameter configuration as presented in Table 6.4, with the lag dimension varied.

6.4 Convolutional auto-encoder

This section will discuss optimal CAE model configurations for both raw MTS- and engineered data. Both raw MTS- and feature engineered performance are discussed to shed light on the superior performance observed for the CAE model on raw MTS data in Figure 6.1. Table 6.5 presents the optimal CAE model configuration identified for both data types.

Table 6.5: Summary of optimal design parameters for CAE FPR models

Design parameter	**Investigated levels**	**Raw MTS**	**Engineered data**
CAE architecture	Short network (Figure 5.9, right) Long network (Figure 5.9, left)	Long architecture	Short architecture
Recognition window length, l_w	$0, 30, \ldots, 330, 360$	210	120
CAE model FPR performance			
Dataset	$AUC_{ROC,\psi_{min}}$, raw MTS data		$AUC_{ROC,\psi_{min}}$, feature engineered data
Train	0.487		0.528
Validation	0.408		0.497
Test	0.419		0.488

Table 6.5 shows that the optimal CAE model applied to raw MTS data is slightly over-fitted to the training data, as the $AUC_{ROC,\psi_{min}}$-value observed for the raw MTS model decreases from 0.487 on the training data to 0.419 on the testing data. However, the difference between $AUC_{ROC,\psi_{min}}$-values observed on validation- and testing data (0.408 and 0.419, respectively) is smaller, suggesting that the CAE model

applied to raw MTS data does generalize beyond the training set. Figure 6.21 illustrates how both the optimal CAE models generalize:

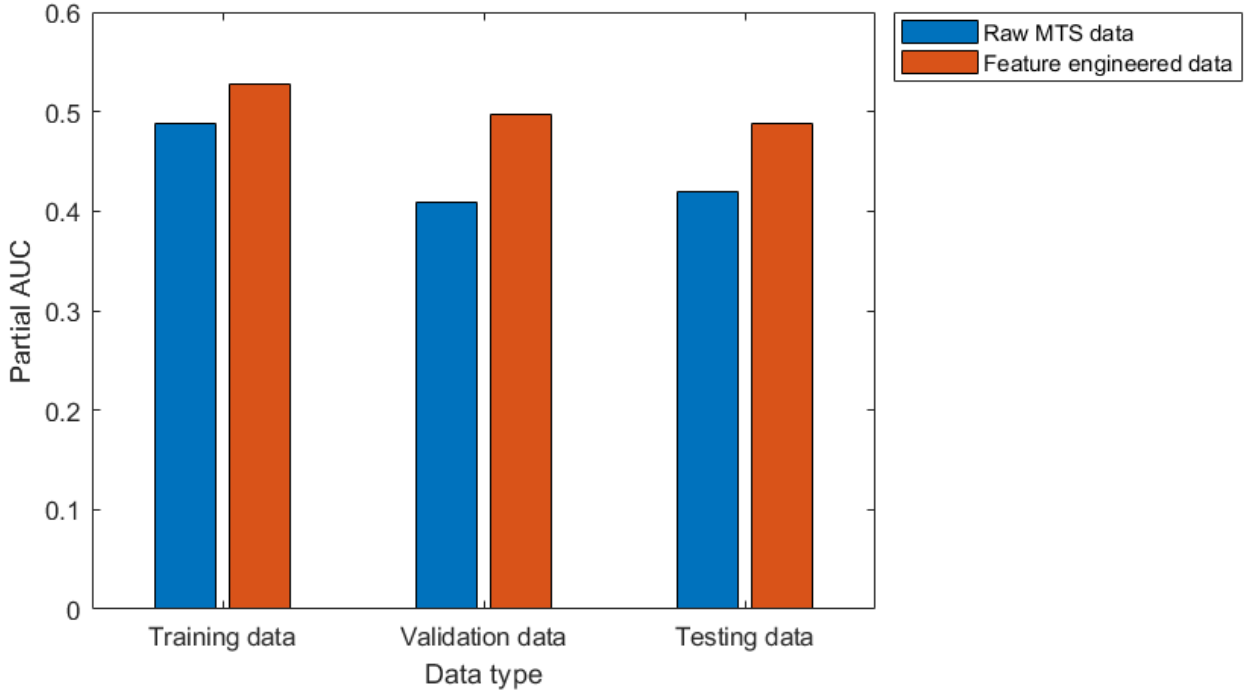

Figure 6.21: Illustration of the optimal CAE models' generalizability, applied to both raw MTS data and feature engineered data.

Table 6.5 shows that evaluating discriminant values in a moving window yields the optimal CAE model configurations. Figure 6.22 shows how FPR performance varies for each model with recognition window length.

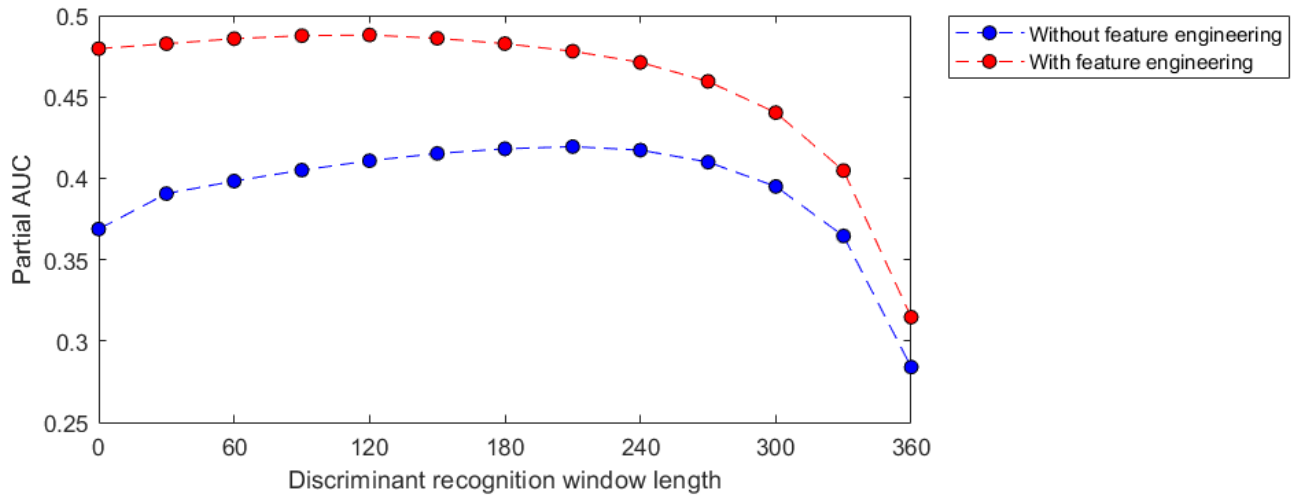

Figure 6.22: Illustration of $AUC_{ROC,\,\psi_{min}}$-values obtained for the optimal CAE configurations in Table 6.5 with varying recognition window length. The red markers show the $AUC_{ROC,\,\psi_{min}}$-value obtained on the feature engineered data at each recognition window length investigated, while the blue markers show the $AUC_{ROC,\,\psi_{min}}$-values obtained on the raw MTS data.

Figure 6.22 shows that evaluating discriminant values in a window has a prominent effect on the FPR performance of the CAE model applied to raw MTS data; the $AUC_{ROC,\psi_{min}}$-value increases from 0.369 to 0.419 at $l_w = 210$. However, window evaluation has a muted effect when applying the CAE model to feature engineered data; evaluating discriminant values in a window with a length $l_w = 120$ only increases the $AUC_{ROC,\psi_{min}}$-value from 0.480 to 0.488. This suggests that the CAE model applied to raw MTS data is plagued by individual discriminant values exceeding recognition thresholds and causing false positives, however, the CAE model applied to feature engineered data generates discriminant values that can effectively distinguish between faulty- and fault free observations without individual discriminant values causing false positives.

Figure 6.23 illustrates each architecture's FPR performance using ROC-curves, applied to raw MTS data- and feature engineered data. Figure 6.23 shows that the long CAE architecture achieves higher sensitivities than the short CAE architecture on the raw MTS data. The two architectures are distinguished by the extra one-dimensional convolutional layer in the longer architecture. This suggests that the one-dimensional convolutional operation used across the time dimension (also shown in Figure 5.9) effectively extracts dynamic characteristics from MTS data, yielding higher sensitivities.

Figure 6.23 shows that the difference in FPR performance between long- and short CAE architectures is minimal when applied to the feature engineered data. This reveals that the one-dimensional convolution operation did not improve FPR performance on static representations of MTS data, but it also did not significantly reduce sensitivity; the long CAE achieved an $AUC_{ROC,\psi_{min}}$-value of 0.482 on the engineered data, compared to the short CAE achieving an $AUC_{ROC,\psi_{min}}$-value of 0.488.

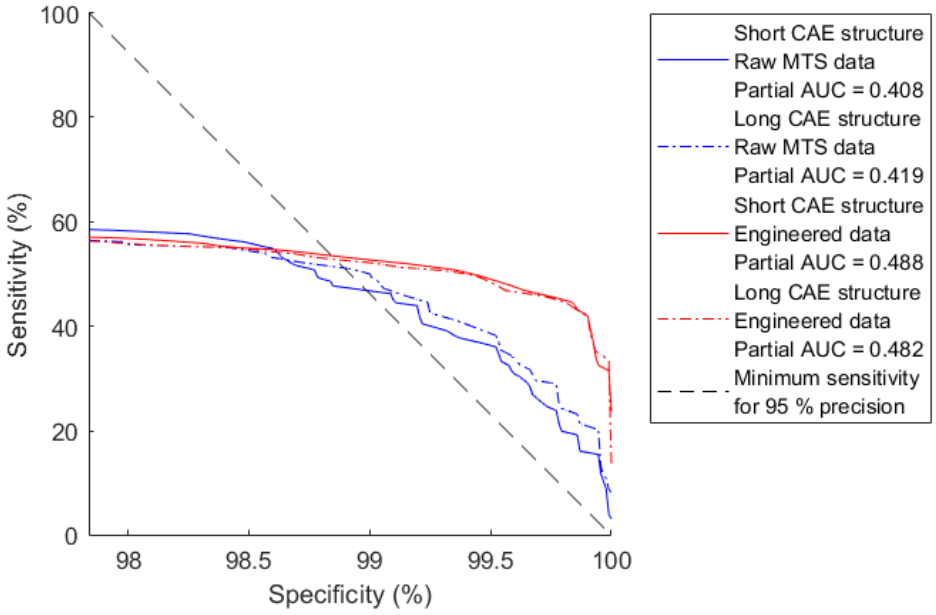

Figure 6.23: ROC-curve of the CAE models applied to raw MTS data (blue lines) and feature engineered data (red lines). Solid lines refer to sensitivities obtained for the short CAE structure, dashed-and-dotted lines refer to sensitivities obtained using the long CAE structure.

6.5 Optimal FPR model demonstration

This section delves deeper into the performance of the optimal AE-model, applied to feature engineered data, and the optimal CAE-model, applied to raw MTS data. The first of these models are selected for evaluation because it achieved the highest $AUC_{ROC,\psi_{min}}$-value of all investigated model configurations. These second of these is selected because its $AUC_{ROC,\psi_{min}}$-value is far higher compared to other model types when applied to raw MTS data. This section considers the metrics of FPR performance given in Table 6.6 for each of these models:

Table 6.6: FPR performance metrics

FPR performance metric		Description
1.	Average detection delay	Shows how long blowback-preceding conditions are present before they are recognized by the model.
2.	Mean warning period before blowback occurs	Shows how much time is available for corrective actions before a blowback occurs.
3.	Specificity	Indicates how many false alarms occurs at a specific recognition threshold.
4.	Number of missed blowback predictions	Indicates how many blowbacks are not predicted by the model at a specific recognition threshold.

Each model is compared for three recognition threshold levels. These thresholds are selected according to Table 6.7. Table 6.7 also provides motivation for evaluating the models at the given thresholds.

Table 6.7: Recognition thresholds selected for evaluation

Recognition threshold		Motivation
1.	Threshold where 95 % precision is achieved.	This is the minimum desired precision defined for the FPR model, and is used to calculate the $AUC_{ROC,\psi_{min}}$-value in section 5.4.
2.	Maximum threshold where no blowbacks are not predicted.	Obviously predicting each blowback is desirable, therefore evaluating performance at the threshold where this is possible is desired.
3.	Minimum threshold where 100 % specificity is achieved.	At this threshold, the model would yield no false alarms. Therefore it is useful to evaluate performance here.

The optimal auto-encoder configuration presented in Table 6.4 is applied to feature engineered data in the testing set. The discriminant values it generates are illustrated in Figure 6.24, along with the recognition thresholds defined in Table 6.7. The long-structure CAE model presented in Table 6.5 is applied to raw MTS data in the testing set, and the resulting discriminant values are presented in Figure 6.25.

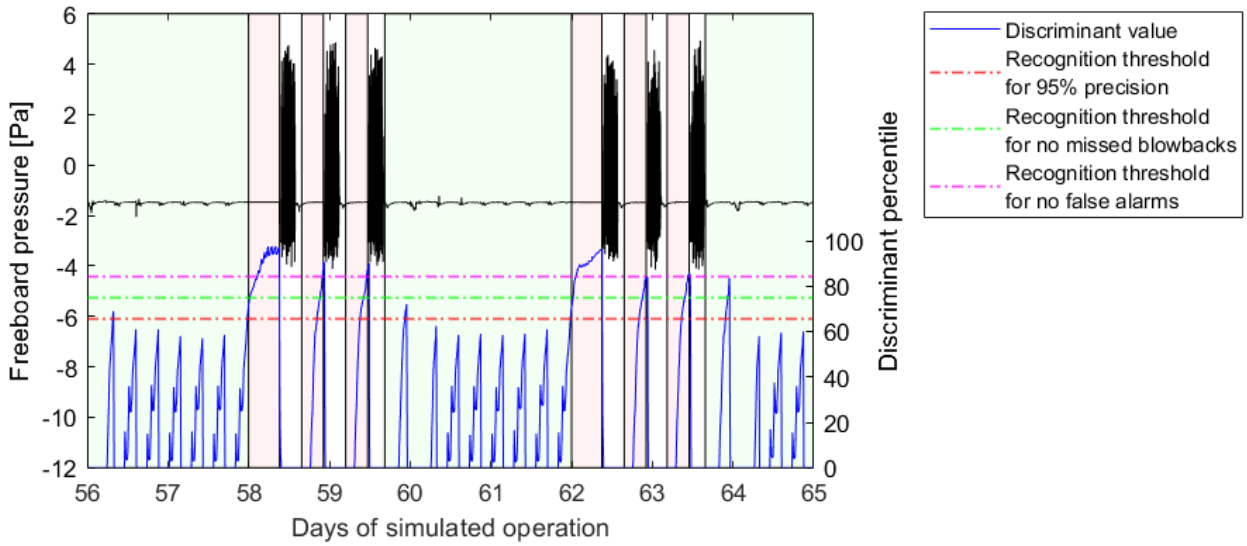

Figure 6.24: Illustration of AE-fault pattern recognition for 9 days of simulated operation. The areas shaded in green indicate fault-free observations, red areas indicate blowback-preceding observations. The blue line is the discriminant value generated by the AE model, and the solid black line indicates freeboard pressure. Dashed horizontal lines correspond to the different recognition thresholds defined in Table 6.7.

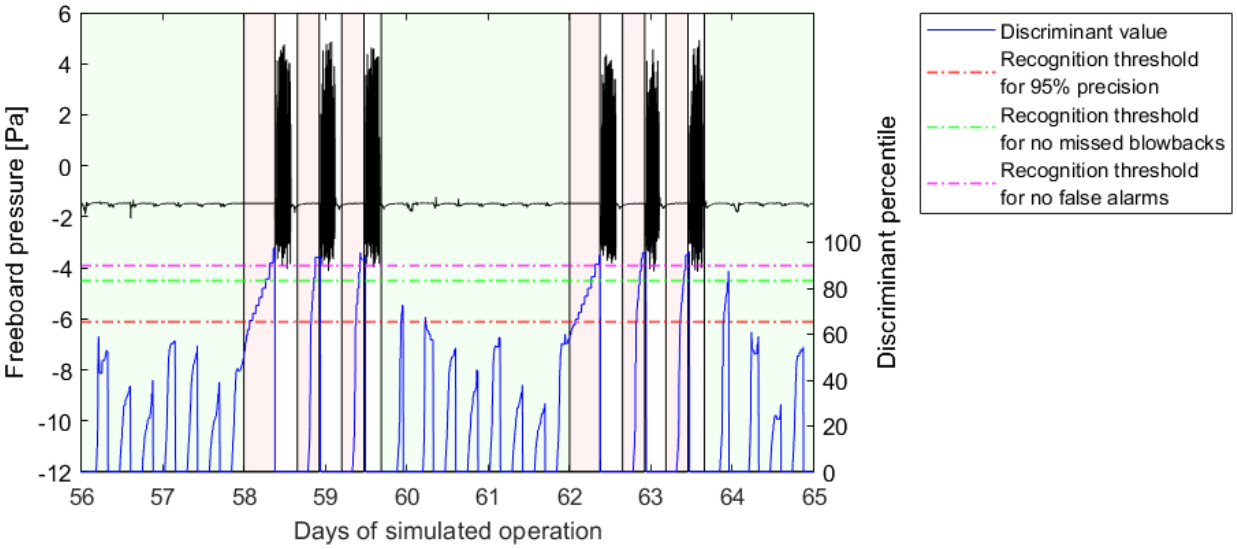

Figure 6.25: Illustration of CAE-fault pattern recognition for 9 days of simulated operation. The areas shaded in green indicate fault-free observations, red areas indicate blowback-preceding observations. The blue line is the discriminant value generated by the CAE model, and the solid black line indicates freeboard pressure. Dashed horizontal lines correspond to the different recognition thresholds defined in Table 6.7.

Figure 6.24 and Figure 6.25 shows that the recognition thresholds for no false alarms (dashed magenta line) falls above the recognition threshold for no missed blowbacks (dashed green line). This shows that the AE-FPR model applied to feature engineered data and the CAE-FPR model applied to raw MTS data are both unable to predict each blowback in the testing set without causing any false alarms.

Table 6.8: FPR performance metrics observed at the recognition threshold required for 95% precision

FPR performance metric		AE-FPR model	CAE-FPR model
1.	Average detection delay	$3.4\ h$	$3.7\ h$
2.	Mean warning period before blowback occurs	$4.8\ h$	$4.5\ h$
3.	Specificity	$98.9\ \%$	$98.8\ \%$
4.	Missed blowback predictions	0	0

Table 6.8 shows that neither the AE- nor CAE-FPR models fail to predict furnace blowbacks at the recognition threshold required for 95 % precision. However, the AE-FPR model recognizes blowbacks approximately 18 minutes sooner than the CAE-FPR model, at higher specificities.

Table 6.9: FPR performance metrics observed at the recognition threshold required to predict each blowback

FPR performance metric		AE-FPR model	CAE-FPR model
1.	Average detection delay	$4.5\ h$	$6.0\ h$
2.	Mean warning period before blowback occurs	$3.7\ h$	$2.2\ h$
3.	Specificity	$99.87\ \%$	$99.95\ \%$
4.	Missed blowback predictions	0	0

Table 6.9 shows that both the AE- and CAE-FPR models are able to predict all blowbacks. However, the AE-FPR model would provide an additional 90 minutes of warning to operators compared to the CAE-FPR model.

Table 6.10: FPR performance metrics observed at the recognition threshold required for no false positives

FPR performance metric		AE-FPR model	CAE-FPR model
1.	Average detection delay	$5.5\ h$	$6.9\ h$
2.	Mean warning period before blowback occurs	$2.7\ h$	$1.3\ h$
3.	Specificity	$100\ \%$	$100\ \%$
4.	Missed blowback predictions	12	14

Table 6.10 shows that the AE-FPR model provides more than double the warning time before a blowback occurs compared to the CAE-FPR model. Furthermore, the AE-FPR model fails to predict 12 out of 63 blowbacks, compared to the 14 missed by the CAE-FPR model.

7 CONCLUSIONS AND RECOMMENDATIONS

This work focused on exploring data-driven models for blowback prediction in SAFs. A simple SAF model that emulates blowbacks was developed and presented; this model was used to generate large volumes of dynamic, nonlinear data on which to develop FPR models. This addressed the first objective identified for this project, as the simulated data facilitated objective comparisons of FPR model performance; each model is evaluated based on how well it recognizes blowback-preceding conditions in the simulated data.

Reconstruction-based FPR models were selected for investigation, as they can be trained on data with labelling constraints typical of industrial processes. Simple feature engineering techniques were selected and applied to the simulated data. These feature engineering techniques attempted to provide more informative representations of MTS for model development, yielding engineered data that is more suitable for FPR. As part of this investigation, algorithms for developing and implementing reconstruction-based FPR models to process data were presented. This addressed the second objective identified for this project.

This investigation considered PCA, kernel PCA, auto-encoders and one dimensional convolutional CAEs as FPR models. The application of one dimensional CAEs to MTS as reconstruction-based FPR models is novel compared to the other, more established FPR models investigated. Two one dimensional CAE architectures were developed for the MTS data generated by the SAF model. Other FPR model types attempted to model dynamic behaviour by lagging modelled data.

Each of the above FPR models were applied to the MTS data generated by the SAF model and thoroughly compared. Partial ROC curves were used to quantify recognition performance above 95 % precision, facilitating objective model comparisons and addressing the third objective identified for this project. Section 7.1 briefly recaps the furnace model developed to meet the project aim. Conclusions drawn from FPR model comparisons are presented in sections 7.2 to 7.6. Sections 7.7 and 7.8 gives summarizing recommendations on applying FPR models and expanding the work presented in this thesis.

7.1 Developed submerged arc furnace model

An SAF model has been developed and presented. The developed model approximates the SAF interior as a set of ordinary differential equations, derived from mass- and energy balances over distinct furnace zones. This model is distinguished from previous approaches to modelling SAFs by being simple enough to generate potentially weeks of simulated data, while still simulating dynamic furnace behaviour. The developed model was crucial to this project, as it provided simulated furnace data containing blowbacks where the causes of blowbacks were known. Furthermore, this furnace model simulated changes in cooling water flowrate, concentrate feed composition and extraction draught pressure. These changing operating conditions facilitated more objective FPR model comparisons, as developed FPR models had to recognize blowback-preceding conditions from different process behaviours.

7.2 Linear PCA versus kernel PCA

A consensus in the literature consulted for this investigation exists that kernel PCA is superior to standard linear PCA in process monitoring applications. These sources emphasized kernel PCA's ability to efficiently extract relevant features from infinite dimensional nonlinear feature spaces as a key strength over standard PCA for online process monitoring.

This investigation confirmed that kernel PCA is superior to linear PCA when applied on a small dataset, but found that kernel PCA's superior performance is not maintained when applied to larger datasets. Kernel PCA computational complexity grows exponentially with dataset size, requiring that data be approximated in a low dimension. The k-means approximation used in this investigation allowed kernel PCA to be used on the large simulated dataset, but its performance is inferior to standard PCA applied to the entire dataset.

The results obtained in this investigation highlighted a key strength of linear PCA over kernel PCA that only becomes prominent in large datasets; linear PCA models' performance increases the more data is available to train them. In contrast, a kernel PCA model's performance is constrained by how well a large dataset can be approximated. This presents a significant challenge to scaling kernel PCA models to industrial data characterized by massive volumes of observations collected over many years.

7.3 Nonlinear versus linear models

This investigation found that nonlinear FPR models outperform linear models (excluding kernel PCA, for reasons given in the previous section). The one dimensional CAE and AE FPR models outperformed linear PCA, suggesting that processes with nonlinear characteristics are best monitored by nonlinear process models.

However, the algorithms for developing and applying linear PCA models for FPR are far simpler than their counterparts for one dimensional CAE and AE FPR models. Only the number of principal components to retain and the lag dimension need to be defined when deriving a PCA model, and a globally optimized model is calculated immediately. This is in contrast to the one dimensional CAE and AE models, where network architectures need to be constructed manually and optimized with potentially time-consuming algorithms.

7.4 Comparison of one dimensional CAE and AE models

The one dimensional CAE FPR models developed in this investigation outperformed the less sophisticated AE FPR models when applied to the raw simulated MTS data, and by a significant margin. This confirms that one-dimensional convolutional layers are superior to fully connected layers in learning features from MTS. Furthermore, the one dimensional CAE models could recognize all blowback-preceding conditions with a minimum precision of 95 % in the raw simulated MTS data.

Applying feature engineering to the simulated SAF data greatly narrowed the performance gap between the one dimensional CAE and AE models. The results obtained in this investigation showed that simple AE models outperform one dimensional CAE models when applied to the engineered simulated data (see

section 6.5). This shows that a one dimensional CAE architecture designed for extracting features from MTS would not be guaranteed to maintain superior performance on features engineered from MTS.

7.5 Role of feature engineering in FPR

This investigation found that the choice of FPR model type (PCA, kernel PCA, AE or one dimensional CAE) has a prominent impact on model performance, but that this impact is overshadowed by the manual feature engineering techniques applied. Manual feature engineering more than doubled the FPR performance of the PCA-, kernel PCA- and AE FPR models evaluated, and significantly boosted the performance of one dimensional CAE models.

This suggests that FPR model type selection should have a lower priority than selecting proper MTS feature engineering techniques; the optimal model type in this investigation depends on whether or not feature engineering was employed. The prominent role that feature engineering plays in FPR applications favour using simpler FPR models; simpler models are faster to train and implement, allowing different feature engineering techniques to be applied and evaluated.

However, the feature engineering techniques used in this project were selected to replace dynamic variations with static representations. The superior performance of the one dimensional CAE models on the raw MTS data suggests that one dimensional CAE models would perform better if feature engineering techniques that conserve dynamic variations were employed.

7.6 Modelling dynamic characteristics

The data generated by the developed SAF model contained dynamic characteristics. The linear PCA-, kernel PCA- and AE FPR approaches modelled process dynamics by lagging observations with previous observations. This investigation found that these models are unable to extract effective features from lagged data; linear PCA and AE FPR models displayed optimal performance on unlagged data, and dynamic characteristics are better represented by moving window statistics than lagged values. The one-dimensional convolution operation used by the CAE models in this project were superior in modelling dynamic characteristics.

7.7 Recommendations for applying FPR models

This section presents recommendations for developing and applying FPR models based on the conclusions presented in sections 7.2 to 7.6.

1. Sophisticated approximation algorithms are required to implement Kernel PCA on large data volumes. Standard linear PCA should be used if an effective low dimensional representation of a large dataset cannot be constructed.

2. Feature engineering should be implemented and optimized before selecting and developing sophisticated FPR models, because superior performance on raw data does not guarantee superior performance on engineered data.

3. Nonlinear AE- and one dimensional CAE models offer superior FPR performance to linear PCA models, but linear PCA models are far simpler and faster to implement. This suggests that they are ideal for discovering optimal engineered features, upon which nonlinear models could be developed.

4. Lagging data is not an effective way of incorporating dynamic characteristics in observations. The one-dimensional convolution approach used in this project yielded far superior results when modelling blowback-preceding conditions.

5. One dimensional CAE models are superior to fully connected AE models when applied to MTS data, but AE models are more effective when presented with informative, representative features generated from MTS. Fully connected AE models applied to features engineered from MTS offer the best performance of the FPR models evaluated in this investigation.

7.8 Recommendations for further investigation

This section presents aspects of this investigation that could be expanded on in future work.

1. The furnace model developed for this project only had one blowback generating mechanism. This model could be expanded to simulate multiple causes of blowbacks, allowing the presented FPR approaches to be evaluated on different blowback causes.

2. The furnace model developed for this project allows large volumes of dynamic process data to be generated, and different process faults can be simulated easily. It is not limited to only simulating blowbacks. This model is therefore suitable as a benchmark process for generating data for evaluating different FPR approaches.

3. Approximation algorithms can be investigated with which kernel PCA can be extended to large datasets. This investigation used k-means clustering as approximation algorithm, but alternative approximation approaches may yield superior performance.

4. The presented FPR algorithms have not been tested on industrial data. The model development- and implementation algorithms presented in sections 5.5 to 5.8 are suitable for and can be translated to industrial data sets, but quantifying model performance as in section 5.4 requires a labelled testing set. A future investigation into quantifying model performance on unlabelled data will help in extending the presented FPR models to industrial data.

8 REFERENCES

Ammiche, M., Kouadri, A., Bensmail, A., 2018. A Modified Moving Window dynamic PCA with Fuzzy Logic Filter and application to fault detection. Chemom. Intell. Lab. Syst. 177, 100–113. https://doi.org/10.1016/j.chemolab.2018.04.012

Basha, S.H.S., Dubey, S.R., Pulabaigari, V., Mukherjee, S., 2020. Impact of fully connected layers on performance of convolutional neural networks for image classification. Neurocomputing 378, 112–119. https://doi.org/10.1016/j.neucom.2019.10.008

Bekker, J.G., Craig, I.K., Pistorius, P.C., 2000. Model predictive control of an electric arc furnace off-gas process. Control Eng. Pract. 8, 445–455.

Bezuidenhout, J.J., Eksteen, J.J., Bradshaw, S.M., 2009. Computational fluid dynamic modelling of an electric furnace used in the smelting of PGM containing concentrates. Miner. Eng. 22, 995–1006. https://doi.org/10.1016/j.mineng.2009.03.009

Bishop, C., 2006. Pattern recognition and machine learning, Springer. https://doi.org/10.1128/aac.03728-14

Calli, E., 2017. Master's Thesis Faster Convolutional Neural Networks.

Carbone, A., 2009. Detrending moving average algorithm: A brief review. TIC-STH'09 2009 IEEE Toronto Int. Conf. - Sci. Technol. Humanit. 691–696. https://doi.org/10.1109/TIC-STH.2009.5444412

Charte, D., Charte, F., del Jesus, M.J., Herrera, F., 2020. An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges. Neurocomputing 404, 93–107. https://doi.org/10.1016/j.neucom.2020.04.057

Chen, J., Liao, C., 2002. Dynamic process fault monitoring based on neural network and PCA 12, 277–289.

Chen, S., Yu, J., Wang, S., 2020. One-dimensional convolutional auto-encoder-based feature learning for fault diagnosis of multivariate processes. J. Process Control 87, 54–67. https://doi.org/10.1016/j.jprocont.2020.01.004

Cheng, P., Li, W., Ogunbona, P., 2009. Greedy approximation of kernel PCA by minimizing the mapping error. DICTA 2009 - Digit. Image Comput. Tech. Appl. 303–308. https://doi.org/10.1109/DICTA.2009.57

Choi, S.W., Lee, I.B., 2004. Nonlinear dynamic process monitoring based on dynamic kernel PCA. Chem. Eng. Sci. 59, 5897–5908. https://doi.org/10.1016/j.ces.2004.07.019

Cramer, L.A., 2001. The extractive metallurgy of South Africa's platinum ores. JOM 53, 14–18. https://doi.org/10.1007/s11837-001-0048-1

Crundwell, F.K., Moats, M.S., Ramachandran, V., Robinson, T.G., Davenport, W.G., 2011a. Smelting and Converting of Sulfide Concentrates Containing Platinum-Group Metals. Extr. Metall. Nickel, Cobalt Platin. Gr. Met. 437–456. https://doi.org/10.1016/b978-0-08-096809-4.10035-8

Crundwell, F.K., Moats, M.S., Ramachandran, V., Robinson, T.G., Davenport, W.G., 2011b. Smelting Laterite Concentrates to Sulfide Matte. Extr. Metall. Nickel, Cobalt Platin. Gr. Met. 95–107. https://doi.org/10.1016/b978-0-08-096809-4.10008-5

Cybenko, G., 1989. Approximation by Superpositions of a Sigmoidal Function. Math. Control. Signals, Syst. 2, 303–314. https://doi.org/10.1007/BF02551274

Deng, X., Tian, X., 2013. Nonlinear process fault pattern recognition using statistics kernel PCA similarity factor. Neurocomputing 121, 298–308. https://doi.org/10.1016/j.neucom.2013.04.042

Dodd, L.E., Pepe, M.S., 2003. Partial AUC estimation and regression. Biometrics 59, 614–623. https://doi.org/10.1111/1541-0420.00071

Dong, D., Mcavoy, T.J., 1996. Nonlinear principal component analysis - Based on principal curves and neural networks. Comput. Chem. Eng. 20, 65–78. https://doi.org/10.1016/0098-1354(95)00003-K

Dong, Y., Qin, S.J., 2018. A novel dynamic PCA algorithm for dynamic data modeling and process monitoring. J. Process Control 67, 1–11. https://doi.org/10.1016/j.jprocont.2017.05.002

Eksteen, J.J., 2011. A mechanistic model to predict matte temperatures during the smelting of UG2-rich blends of platinum group metal concentrates. Miner. Eng. 24, 676–687. https://doi.org/10.1016/j.mineng.2010.10.017

Eksteen, J.J., Van Beek, B., Bezuidenhout, G.A., 2011. Cracking a hard nut: An overview of Lonmin's operations directed at smelting of UG2-rich concentrate blends. J. South. African Inst. Min. Metall. 111, 681–690.

Ge, Z., Yang, C., Song, Z., 2009. Improved kernel PCA-based monitoring approach for nonlinear processes. Chem. Eng. Sci. 64, 2245–2255. https://doi.org/10.1016/j.ces.2009.01.050

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep Sparse Rectifier Neural Networks, in: International Conference on Artificial Intelligence and Statistics. pp. 315–323. https://doi.org/10.1002/ecs2.1832

Gredilla, A., Fdez-Ortiz de Vallejuelo, S., de Diego, A., Madariaga, J.M., Amigo, J.M., 2013. Unsupervised pattern-recognition techniques to investigate metal pollution in estuaries. TrAC - Trends Anal. Chem. 46, 59–69. https://doi.org/10.1016/j.trac.2013.01.014

Groenewald, J.W.D., Nelson, L.R., Hundermark, R.J., Phage, K., Sakaran, R.L., Van Rooyen, Q., Cizek, A., 2018. Furnace integrity monitoring using principal component analysis: An industrial case study. J. South. African Inst. Min. Metall. 118, 345–352. https://doi.org/10.17159/2411-9717/2018/v118n4a3

He, L., Zhang, H., 2018. Kernel K-Means Sampling for Nyström Approximation 27, 2108–2120.

Hu, Y., Ping, B., Zeng, D., Niu, Y., Gao, Y., Zhang, D., 2020. Research on fault diagnosis of coal mill system based on the simulated typical fault samples. Measurement 161, 107864. https://doi.org/10.1016/j.measurement.2020.107864

Hundermark, R., de Villiers, B., Ndlovu, J., 2006. Process Description and Short History of Polokwane Smelter. South. African Pyrometallurgy 2006 35–42.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A., 2019. Deep learning for time series classification: a review. Data Min. Knowl. Discov. 33, 917–963. https://doi.org/10.1007/s10618-019-00619-1

Jain, A.K., Duin, R.P.W., Mao, J., 2000. Statistical pattern recognition: A review. IEEE Trans. Pattern Anal. Mach. Intell. 22, 4–37. https://doi.org/10.1109/34.824819

James, G., Witten, D., Hastie, T., Tibshirani, R., 2012. An Introduction to Statistical Learning, Springer. https://doi.org/10.2174/0929867003374372

Jiang, G., Xie, P., He, H., Yan, J., 2018. Wind Turbine Fault Detection Using a Denoising Autoencoder with Temporal Information. IEEE/ASME Trans. Mechatronics 23, 89–100. https://doi.org/10.1109/TMECH.2017.2759301

Jones, R.T., 2005. An overview of South African PGM smelting. Nickel Cobalt 2005 Challenges Extr. Prod. 147–178.

Kassidas, A., Taylor, P.A., MacGregor, J.F., 1998. Off-line diagnosis of deterministic faults in continuous dynamic multivariable processes using speech recognition methods. J. Process Control 8, 381–393. https://doi.org/10.1016/s0959-1524(98)00025-0

Keerthi, S.S., Lin, C.J., 2003. Asymptotic behaviors of support vector machines with gaussian kernel. Neural Comput. 15, 1667–1689. https://doi.org/10.1162/089976603321891855

Khediri, I. Ben, Limam, M., Weihs, C., 2011. Variable window adaptive Kernel Principal Component Analysis for nonlinear nonstationary process monitoring. Comput. Ind. Eng. 61, 437–446. https://doi.org/10.1016/j.cie.2011.02.014

Ko, T., Kim, H., 2020. Fault Classification in High-Dimensional Complex Processes Using Semi-Supervised Deep Convolutional Generative Models. IEEE Trans. Ind. Informatics 16, 2868–2877. https://doi.org/10.1109/TII.2019.2941486

Kramer, M.A., 1991. Nonlinear principal component analysis using autoassociative neural networks. AIChE J. 37, 233–243. https://doi.org/10.1002/aic.690370209

Ku, W., Storer, R.H., Georgakis, C., 1995. Disturbance detection and isolation by dynamic principal component analysis. Chemom. Intell. Lab. Syst. 30, 179–196. https://doi.org/10.1016/0169-7439(95)00076-3

Kubben, P., Dumontier, M., Dekker, A., 2019. Fundamentals of Clinical Data Science. Springer US.

Ledesma, R.D., Valero-Mora, P., Macbeth, G., 2015. The Scree Test and the Number of Factors: a Dynamic Graphics Approach. Span. J. Psychol. 18, E11. https://doi.org/10.1017/sjp.2015.13

Lee, J.M., Yoo, C., Choi, S.W., Vanrolleghem, P., Lee, I.-B., 2004. Nonlinear process monitoring using kernel principal component analysis. Chem. Eng. Sci. 59, 223–234. https://doi.org/10.1109/ICSAI.2012.6223652

Li, C.C., Jeng, J.C., 2010. Multiple sensor fault diagnosis for dynamic processes. ISA Trans. 49, 415–432. https://doi.org/10.1016/j.isatra.2010.05.001

Li, F.-F., Johnson, J., Yeung, S., 2017. Lecture 5 : Convolutional Neural Networks. pp. 1–78.

Liu, X., Li, K., McAfee, M., Irwin, G.W., 2011. Improved nonlinear PCA for process monitoring using support vector data description. J. Process Control 21, 1306–1317. https://doi.org/10.1016/j.jprocont.2011.07.003

Logar, V., Dovzan, D., Skrjanc, I., 2012a. Modeling and Validation of an Electric Arc Furnace: Part 1, Heat and Mass Transfer. Isij Int. 52, 402–412.

Logar, V., Dovzan, D., Skrjanc, I., 2012b. Modeling and Validation of an Electric Arc Furnace: Part 2, Thermo-chemistry. Isij Int. 52, 413–423.

MacGregor, J.F., Kourti, T., 1995. Statistical process control of multivariate processes. Control Eng. Pract. 3, 403–414. https://doi.org/10.1016/0967-0661(95)00014-L

MacRosty, R.D.M., Swartz, C.L.E., 2005. Dynamic modeling of an industrial electric arc furnace. Ind. Eng. Chem. Res. 44, 8067–8083. https://doi.org/10.1021/ie050101b

Mainza, A., Powell, M.S., Knopjes, B., 2005. A comparison of different cyclones in addressing challenges in the classification of the dual density UG2 platinum ore. J. South African Inst. Min. Metall. 105, 341–348.

Mazhelis, O., 2006. One-class classifiers: a review and analysis of suitability in the context of mobile-masquerader detection. South African Comput. J. 36, 29–48.

McGonagle, J., Shaikouski, G., Williams, C., 2020. Backpropagation | Brilliant Math & Science Wiki [WWW Document]. URL https://brilliant.org/wiki/backpropagation/

Merelli, E., Luck, M., 2004. Technical Forum Group on Agents in Bioinformatics, Knowledge Engineering Review. https://doi.org/10.1017/S000000000000000

Misra, M., Yue, H.H., Qin, S.J., Ling, C., 2002. Multivariate process monitoring and fault diagnosis by multi-scale PCA. Comput. Chem. Eng. 26, 1281–1293. https://doi.org/10.1016/S0098-1354(02)00093-5

Mudd, G.M., 2010. Platinum group metals: a unique case study in the sustainability of mineral resources.

4th Int. Platin. Conf. Platin. Transit. 'Boom or Bust' 113–120.

Nell, J., 2004. Melting of platinum group metal concentrates in South Africa. J. South African Inst. Min. Metall. 104, 423–428.

Ng, A., 2017. CS229: Additional Notes on Backpropagation. CS229 Mach. Learn. 1–2.

Ng, A., 2005. CS229 Lecture notes 2 1–14.

Palma, L.B., Ferreira, B.G., Gil, P.S., Coito, F.V., 2015. Fault detection and diagnosis approach based on observers and SVD-PCA. Proc. IEEE Int. Conf. Ind. Technol. 2015-June, 246–251. https://doi.org/10.1109/ICIT.2015.7125106

Pan, Y., Sun, S., Jahanshahi, S., 2011. Mathematical modelling of heat transfer in six-in-line electric furnaces for sulphide smelting. J. South. African Inst. Min. Metall. 111, 717–732.

Ritchie, S., Eksteen, J.J., 2011. Investigating the effect of slag bath conditions on the existence of multiphase emulsion zones in PGM smelting furnaces using computation fluid dynamics. Miner. Eng. 24, 661–675. https://doi.org/10.1016/j.mineng.2010.09.017

Salfner, F., Lenk, M., Malek, M., 2010. A survey of online failure prediction methods. ACM Comput. Surv. 42. https://doi.org/10.1145/1670679.1670680

Sánchez-Fernández, A., Baldán, F.J., Sainz-Palmero, G.I., Benítez, J.M., Fuente, M.J., 2018. Fault detection based on time series modeling and multivariate statistical process control. Chemom. Intell. Lab. Syst. 182, 57–69. https://doi.org/10.1016/j.chemolab.2018.08.003

Schmidhuber, J., 2015. Deep Learning in neural networks: An overview. Neural Networks 61, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Schölkopf, B., Knirsch, P., Smola, A., Burges, C., 1998a. Fast Approximation of Support Vector Kernel Expansions, and an Interpretation of Clustering as Approximation in Feature Spaces 125–132. https://doi.org/10.1007/978-3-642-72282-0_12

Schölkopf, B., Smola, A., Müller, K.-R., 1998b. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. Neural Comput. 10, 1299–1319.

Shashua, A., 2009. Introduction to Machine Learning: Class Notes 67577.

Sheng, Y.Y., Irons, G.A., Tisdale, D.G., 1998a. Transport phenomena in electric smelting of nickel matte: Part I. Electric potential distribution. Metall. Mater. Trans. B Process Metall. Mater. Process. Sci. 29, 77–83. https://doi.org/10.1007/s11663-998-0009-y

Sheng, Y.Y., Irons, G.A., Tisdale, D.G., 1998b. Transport phenomena in electric smelting of nickel matte: Part II. Mathematical modeling. Metall. Mater. Trans. B Process Metall. Mater. Process. Sci. 29, 85–94. https://doi.org/10.1007/s11663-998-0010-5

Shyu, M.L., Chen, S.C., Sarinnapakorn, K., Chang, L., 2003. A Novel Anomaly Detection Scheme Based on Principal Component Classifier. 3rd IEEE Int. Conf. Data Min. 353–365. https://doi.org/10.1007/11539827-18

Singhal, A., Seborg, D.E., 2002. Pattern Matching in Historical Batch Data Using PCA. IEEE Control Syst. 22, 53–63. https://doi.org/10.1109/MCS.2002.1035217

Singhal, A., Seborg, D.E., 2000. Dynamic data rectification using the expectation maximization algorithm. AIChE J. 46, 1556–1565. https://doi.org/10.1002/aic.690460808

Susto, G.A., Cenedese, A., Terzi, M., 2018. Time-Series Classification Methods: Review and Applications to Power Systems Data, Big Data Application in Power Systems. https://doi.org/10.1016/B978-0-12-811968-6.00009-7

Tax, D.M.J., 2001. One-class classification; Concept-learning in the absence of counter-examples. Delft

Univ. Technol. 202. https://doi.org/10.1063/1.3605545

Thethwayo, B., 2010. Sulphidation of Copper Coolers in PGM Smelters.

Trovero, M.A., Leonard, M.J., 2018. Time Series Feature Extraction 2020–2018.

Van Manen, P.K., 2009. Furnace energy efficiency at Polokwane Smelter. J. South. African Inst. Min. Metall. 109, 47–52.

Villalba, S.D., Cunningham, P., 2007. An evaluation of dimension reduction techniques for one-class classification. Artif. Intell. Rev. 27, 273–294. https://doi.org/10.1007/s10462-008-9082-5

Wang, F., Ma, Q., Liu, W., Chang, S., Wang, H., He, J., Huang, Q., 2019. A novel ECG signal compression method using spindle convolutional auto-encoder. Comput. Methods Programs Biomed. 175, 139–150. https://doi.org/10.1016/j.cmpb.2019.03.019

Wang, Q., 2012. Kernel Principal Component Analysis and its Applications in Face Recognition and Active Shape Models.

Westerhuis, J.A., Gurden, S.P., Smilde, A.K., 2000. Generalized contribution plots in multivariate statistical process monitoring. Chemom. Intell. Lab. Syst. 51, 95–114. https://doi.org/10.1016/S0169-7439(00)00062-9

Wise, B.M., Ricker, N.L., Veltkamp, D.F., Kowalski, B.R., 1990. Theoretical basis for the use of principal component models for monitoring multivariate processes. Process Control Qual. 1, 41–51.

Yin, S., Ding, S.X., Naik, A., Deng, P., Haghani, A., 2010. On PCA-based fault diagnosis techniques. Conf. Control Fault-Tolerant Syst. SysTol'10 - Final Progr. B. Abstr. 179–184. https://doi.org/10.1109/SYSTOL.2010.5676031

Zhang, K., Kwok, J.T., 2010. Clustered Nyström method for large scale manifold learning and dimension reduction. IEEE Trans. Neural Networks 21, 1576–1587. https://doi.org/10.1109/TNN.2010.2064786

Zhang, X., Kou, J., Sun, C., 2019. A comparative study of the thermal decomposition of pyrite under microwave and conventional heating with different temperatures. J. Anal. Appl. Pyrolysis 138, 41–53. https://doi.org/10.1016/j.jaap.2018.12.002

Zhang, Y., 2008. Independent Component Analysis (KICA) and Support Vector Machine (SVM). Society 6961–6971.

Zhang, Y., Li, S., Hu, Z., 2012. Improved multi-scale kernel principal component analysis and its application for fault detection. Chem. Eng. Res. Des. 90, 1271–1280. https://doi.org/10.1016/j.cherd.2011.11.015

Zhang, Z., Jiang, T., Li, S., Yang, Y., 2018. Automated feature learning for nonlinear process monitoring – An approach using stacked denoising autoencoder and k-nearest neighbor rule. J. Process Control 64, 49–61. https://doi.org/10.1016/j.jprocont.2018.02.004

Zhang, Z., Jiang, T., Zhan, C., Yang, Y., 2019. Gaussian feature learning based on variational autoencoder for improving nonlinear process monitoring. J. Process Control 75, 136–155. https://doi.org/10.1016/j.jprocont.2019.01.008

APPENDIX A – ANN TRAINING

Network parameters are a crucial part of ANN models. ANNs are trained by finding the optimal weights and biases (both will be denoted w in this section to simplify mathematical formulations) to minimize some error function. The appealing property of ANNs is that simple algorithms can be applied to automatically update ANN parameters (Ng, 2005). This thesis considers three aspects of ANN training: backpropagation, gradient descent and activation functions.

A.1 Backpropagation

When fitting a model to a dataset, parameters are according to their contribution to an error function. Unfortunately, the interconnected nature of ANNs means that an individual parameter's effect on the error function depends on both preceding and succeeding network parameters. Calculating the error function's gradient for each network parameter would be computationally infeasible for all but the simplest ANNs (McGonagle et al., 2020). Backpropagation is a very efficient way of finding the contributions of network parameters to the error function.

Notation used when describing backpropagation.

Symbol	Definition
$\alpha_j^{(J)}$	Input to node j in layer J
$l_j^{(J)}$	Output from node j in layer J
$w_{j,k}^{(J)}$	Weighted connection to node j in layer J from node k
n_J	Number of nodes in layer J

In an AE, the reconstructed input is computed at the final layer:

$$\bar{\mathbf{x}}_i = g\left(\boldsymbol{\alpha}^{(J)}\right) \qquad [\text{A-1}]$$

$\boldsymbol{\alpha}^{(J)}$ is the vector input to the final layer (layer J) of the AE where data object $\mathbf{x}_i$ is reconstructed. The error function for the AE can be expressed as a function of this input:

$$\mathcal{E} = \frac{1}{N}\sum_{i=1}^{N}\left\|g\left(\boldsymbol{\alpha}^{(J)}\right) - \mathbf{x}_i\right\|^2 \qquad [\text{A-2}]$$

The objective in backpropagation is to find the contributions of network parameters, w, to the total error function, $\mathcal{E}$. The partial derivative of $\mathcal{E}$ w.r.t. a weighted connection to the AE output layer (layer J) is given by equation :

$$\frac{\partial \mathcal{E}}{\partial w_{j,k}^{(J)}} = \frac{\partial \mathcal{E}}{\partial \alpha_j^{(J)}} \cdot \frac{\partial \alpha_j^{(J)}}{\partial w_{j,k}^{(J)}} \qquad [\text{A-3}]$$

Note that the input to node j in layer J, $\alpha_j^{(J)}$, is a function of the inputs to layer $J-1$ and the model parameters connected to node j:

$$\alpha_j^{(J)} = \sum_{k=1}^{n_{J-1}} g\left(\alpha_k^{(J-1)}\right) w_{j,k}^{(J)} \qquad [\text{A-4}]$$

The partial derivative of $\alpha_j^{(J)}$ w.r.t. a weighted connection between node j and node k, $\alpha_j^{(J)}$, is computed as follows:

$$\frac{\partial \alpha_j^{(J)}}{\partial w_{j,k}^{(J)}} = g\left(\alpha_k^{(J-1)}\right) = l_k^{(J-1)} \quad [\text{A-5}]$$

The partial derivative of the total error function w.r.t. the input to a node, α_j, is called the error term:

$$\frac{\partial \mathcal{E}}{\partial \alpha_j} = \delta_j \quad [\text{A-6}]$$

Equations A-5 and A-6 can be substituted into equation A-3 as follows:

$$\frac{\partial \mathcal{E}}{\partial w_{j,k}^{(J)}} = \delta_j^{(J)} \cdot l_k^{(J-1)} \quad [\text{A-7}]$$

$l_k^{(J-1)}$ is computed whenever a data object is passed through the ANN. Therefore, only the error term has to be calculated to find the contribution of $w_{j,k}^{(J)}$ to the total error. The error term for a node in the final layer of an AE-ANN is computed as follows:

$$\delta_j^{(J)} = g'\left(\alpha_j^{(J)}\right) \cdot \left(\bar{x}_{ij} - x_{ij}\right) \quad [\text{A-8}]$$

$\bar{x}_{ij}$ is the reconstructed value of x_{ij}. Equation A-7 can be used in conjunction with equation A-8 to find the contribution of model parameters in the final AE-ANN layer. Unfortunately, ANN performance depends on optimized parameters throughout the network. The partial derivative of the total error w.r.t. a parameter connecting to preceding layers (i.e. layer $J-1$) needs to be computed as well.

$$\delta_j^{(J-1)} = \frac{\partial \mathcal{E}}{\partial \alpha_j^{(J-1)}} = \sum_{k=1}^{n_J} \frac{\partial \mathcal{E}}{\partial \alpha_k^{(J)}} \cdot \frac{\partial \alpha_k^{(J)}}{\partial \alpha_j^{(J-1)}} \quad [\text{A-9}]$$

The partial derivative of $\alpha_j^{(J)}$ w.r.t. an input to node k in layer $J-1$, $\alpha_k^{(J-1)}$, is computed as follows:

$$\frac{\partial \alpha_k^{(J)}}{\partial \alpha_j^{(J-1)}} = g'\left(\alpha_k^{(J-1)}\right) w_{k,j}^{(J)} \quad [\text{A-10}]$$

Substituting the output error term (equation 5-7) and equation A-10 into equation A-7 allows the error term for nodes in the layer preceding the output layer to be computed:

$$\delta_j^{(J-1)} = \sum_{k=1}^{n_J} \delta_k^{(J)} \cdot g'\left(\alpha_k^{(J-1)}\right) w_{k,j}^{(J)} \quad [\text{A-11}]$$

Equation 5-10 shows that the error term for a node in a specific layer can be calculated from the error terms of nodes in the succeeding layers (McGonagle et al., 2020). Combining equations A-7 and A-11 gives a formal definition of the backpropagation algorithm:

$$\frac{\partial \mathcal{E}}{\partial w_{j,k}^{(m)}} = l_k^{(m-1)} \cdot g'\left(\alpha_k^{(m)}\right) \cdot \sum_{k=1}^{n_{m+1}} \delta_k^{(m+1)} \cdot w_{k,j}^{(m+1)} \quad [\text{A-12}]$$

Equation A-12 shows where the term backpropagation comes from: first, a training data object is passed through the ANN, activating nodes and yielding node outputs (l). Then, the error term is calculated for

the final ANN layer, using equation A-8. The error terms are then calculated for nodes in inner layers starting from the final layer.

Applying the backpropagation algorithm for ANN training requires initial different ANN parameters (Ng, 2005). If every ANN parameter is the same, then the initial gradient in the total error function would be the same w.r.t. every ANN parameter. This means that the contributions of different ANN parameters to the total error function would be the same. These contributions have to be different to update and optimize the model.

A.2 Parameter optimization

An AE is trained by passing data objects through the network, reconstructing the input at the output. Through backpropagation, the contributions of parameters in the AE are obtained in the form of the partial derivative of the total error function over each network parameter. This section discusses how these partial derivatives are used to update network parameters. All parameter optimization algorithms described here update parameters according to their partial gradient in the total error function. This is why they are called gradient descent methods (Ng, 2005).

In the simplest parameter optimization approach, each parameter is adjusted by the partial derivative of the total error function evaluated for the entire training dataset (Schmidhuber, 2015):

$$w_{l+1} \coloneqq w_l - \eta \left(\frac{\partial \mathcal{E}(\mathbf{X}_t, \bar{\mathbf{X}}_t)}{\partial w_l} \right) \qquad [\text{A-13}]$$

$\mathbf{X}_t$ is the target dataset and $\bar{\mathbf{X}}_t$ is the reconstructed training set. η is the learning rate parameter; larger values of η increases how much each parameter is updated in each iteration. Note that the gradient descent method presented in equation A-13 evaluates the entire dataset when computing the partial derivative of the total error function, at each iteration. This is computationally intensive for an ANN with many parameters applied on a large dataset, even with the computational efficiency of backpropagation.

The stochastic gradient descent method updates ANN parameters using a randomly selected subset of the training set during each iteration. This reduces the computational intensity of each training iteration:

$$w_{l+1} \coloneqq w_l - \eta \left(\frac{\partial \mathcal{E}\left(\mathbf{X}_0^{(s)}, \bar{\mathbf{X}}_0^{(s)}\right)}{\partial w_l} \right) \qquad [\text{A-14}]$$

The stochastic gradient descent method presented in equation 5-16 is vulnerable to local minima in the partial derivative of the total error function. Adding a momentum term to equation A-14 improves the learning by preventing oscillations around a local minima (Schmidhuber, 2015):

$$w_{l+1} \coloneqq w_l - \eta \left(\frac{\partial \mathcal{E}\left(\mathbf{X}_0^{(s)}, \bar{\mathbf{X}}_0^{(s)}\right)}{\partial w_l} \right) + \gamma (w_l - w_{l-1}) \qquad [\text{A-15}]$$

A.3 Node activation functions

The type of node activation functions selected in an ANN has a great effect on ANN training and performance. This section will only discuss nonlinear activation functions; the universal approximation

ability of ANNs described by Cybenko (1989) assumes that nonlinear functions are used. If a linear activation function is employed, then the resulting ANN will also be a linear model.

Logistic sigmoid and hyperbolic tangent activation functions two widely used nonlinear activation functions in ANNs (Ng, 2005). The equations for these two activation functions are presented below:

$$g(\alpha)_{sigmoid} = \frac{1}{1+e^{-\alpha}} \qquad [\text{A-16}]$$

$$g(\alpha)_{tanh} = \frac{e^{\alpha}-e^{-\alpha}}{e^{\alpha}+e^{-\alpha}} \qquad [\text{A-17}]$$

The outputs of these activation functions, and their respective gradients, are illustrated in Figure A.0.1:

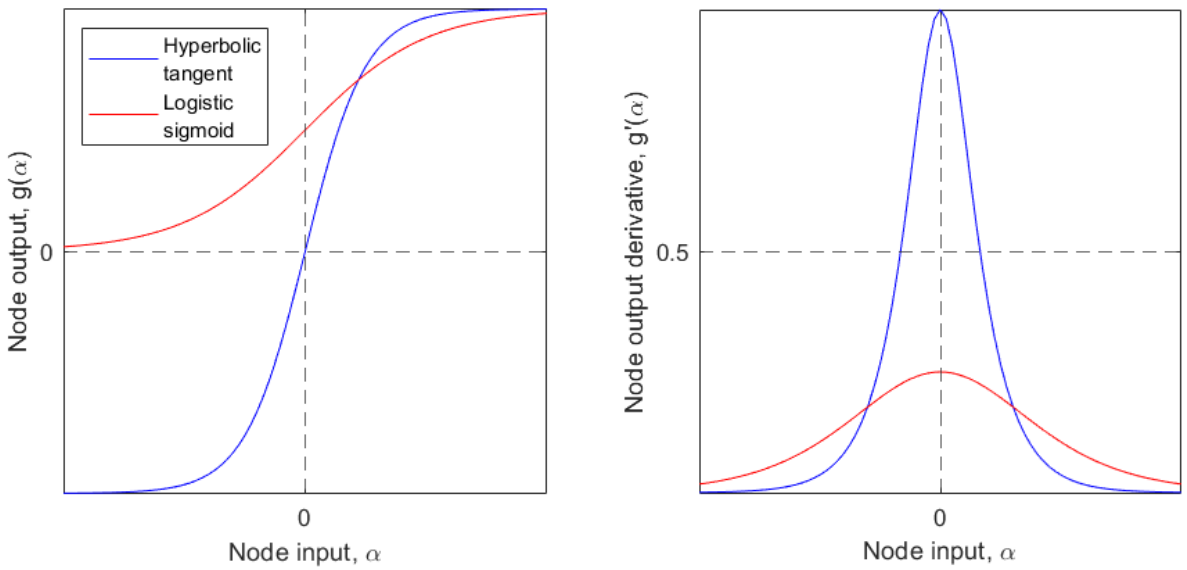

Figure A.0.1: Nonlinear activation function outputs (left) and derivatives (right)

Figure A.0.1 reveals a shortcoming of sigmoidal and tangential activation functions. Gradient descent training methods use the partial derivative of the total error function w.r.t. ANN parameters to optimize the network. The backpropagation algorithm as presented in equation A-12 shows that the gradient of the activation function, $g'(\alpha)$, is used to find the partial derivative of the total error. Figure A.0.1 illustrates that this gradient tends to zero for both kinds of activation function. An implication of this is that ANN parameters are not updated for small or large values of α (Glorot et al., 2011), causing training to effectively stop. This phenomenon is called the vanishing gradient problem: ANN parameters are prevented from being updated due to small activation function gradients.

The ReLU activation function, presented in equation A-18, is used to avoid the vanishing gradient problem in ANN training:

$$g(\alpha)_{ReLU} = \max(\alpha, 0) \qquad [\text{A-18}]$$

The gradient of the ReLU activation function is computed with equation A-19:

$$g'(\alpha)_{ReLU} = \begin{cases} 1 \; if \; \alpha > 0 \\ 0 \; if \; \alpha < 0 \end{cases} \qquad [\text{A-19}]$$

Figure A.0.2 illustrates the ReLU function output- and gradient:

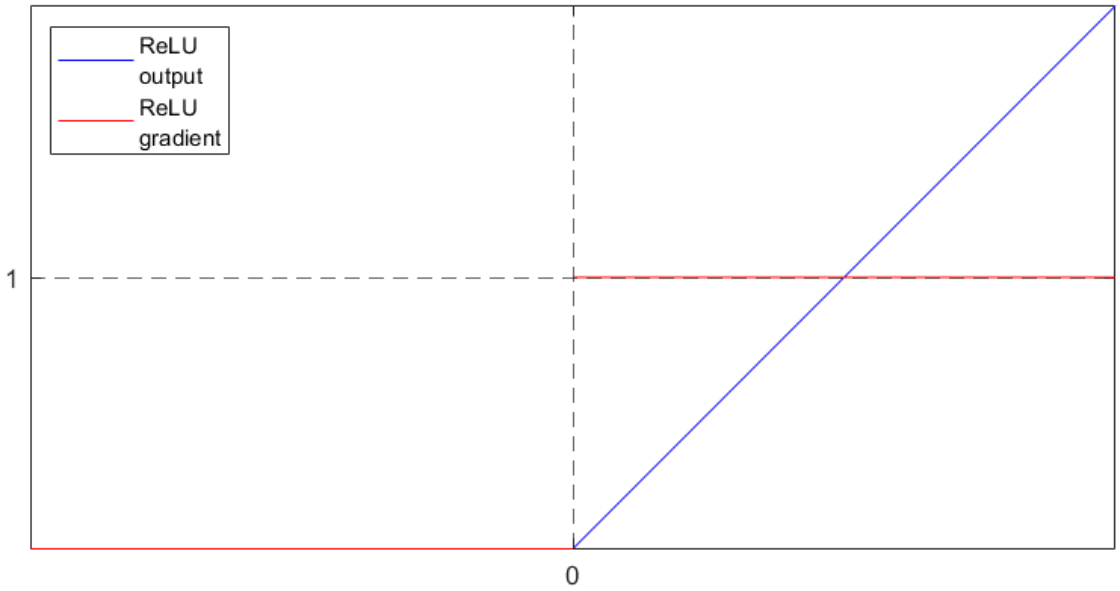

Figure A.0.2: ReLU function output- and gradient

Figure A.0.2 shows how the ReLU function avoids the vanishing gradient problem: the function output is only saturated in one direction (Glorot et al., 2011). Furthermore, the gradient of the ReLU function is simpler to compute for a given input compared to the logistic sigmoid or hyperbolic tangent functions. For a large number of parameters and training iterations, this simpler computation leads to faster network training (Ng, 2017).

APPENDIX B – FURNACE MODEL SYMBOLS AND PARAMETERS

Variable symbols:					
C	Concentration	$\frac{mol}{m^3}$	r_F	Desulphurization reaction rate	$\frac{mol}{m^3 \cdot s}$
F	Molar flow	$\frac{mol}{s}$	r_C	Electrode oxidation rate	$\frac{mol}{s}$
J	Molar flux	$\frac{mol}{m^2 \cdot s}$	R_Ω	Resistivity	$\frac{V^2}{kW}$
L	Length	m	T	Temperature	K
N	Molar amount	mol	V	Volume	m^3
P	Pressure	Pa	v	Volume transfer	$\frac{m^3}{s}$
Q	Heat transfer	kW			
Zone subscripts:					
Bulk concentrate zone		$C(B)$	Reaction gas in concentrate		$C(R)$
Smelting concentrate zone		$C(S)$	Furnace freeboard		G
Slag zone		S	Copper coolers		W
Matte zone		M			
Component subscripts:					
Slag		XO	Reaction gas		R
Matte		XS	Air		A
Sulfurized matte		XS_2			

Molar masses:			
M_G	Freeboard gas molar mass	$29 \cdot 10^{-3}\ kg \cdot mol^{-1}$	Molar mass of air.
M_{XO}	Lumped slag molar mass	$72 \cdot 10^{-3}\ kg \cdot mol^{-1}$	Molar mass of FeO
M_{XS}	Lumped matte molar mass	$88 \cdot 10^{-3}\ kg \cdot mol^{-1}$	Molar mass of FeS
M_{XS_2}	Sulfurized matte molar mass	$120 \cdot 10^{-3}\ kg \cdot mol^{-1}$	Molar mass of FeS_2
Densities:			
ρ_C	Bulk concentrate density	$1600\ kg \cdot m^{-3}$	Eksteen (2011)
ρ_S	Liquid slag density	$2960\ kg \cdot m^{-3}$	Eksteen (2011)

ρ_M	Liquid matte density	$4800\ kg \cdot m^{-3}$	Eksteen (2011)
Heat of fusion			
λ_C	Concentrate heat of fusion	$133\ kJ \cdot mol^{-1}$	Crundwell et al. (2011a)
Heat capacities			
$c_{P,C}$	Concentrate heat capacity	$75 \cdot 10^{-3}\ kJ \cdot mol^{-1} \cdot K^{-1}$	Eksteen (2011)
$c_{P,S}$	Liquid slag heat capacity	$99 \cdot 10^{-3}\ kJ \cdot mol^{-1} \cdot K^{-1}$	Eksteen (2011)
$c_{P,M}$	Liquid matte heat capacity	$78 \cdot 10^{-3}\ kJ \cdot mol^{-1} \cdot K^{-1}$	Eksteen (2011)
$c_{P,G}$	Freeboard heat capacity	$30 \cdot 10^{-3}\ kJ \cdot mol^{-1} \cdot K^{-1}$	Logar et al. (2012a, 2012b)
$c_{P,W}$	Cooling water heat capacity	$75 \cdot 10^{-3}\ kJ \cdot mol^{-1} \cdot K^{-1}$	Logar et al. (2012a, 2012b)
Reaction activation energy			
$E_{A,F}$	Desulphurization reaction activation energy	$150 \cdot 10^{3}\ J \cdot mol^{-1}$	X. Zhang et al. (2019)
$E_{A,C}$	Electrode oxidation activation energy	$120 \cdot 10^{3}\ J \cdot mol^{-1}$	Logar et al. (2012a, 2012b)
Rate constants			
k_V	Concentrate mixing constant	$2 \cdot 10^{-4}\ s^{-1}$	Assumed value*
k_{PR}	Freeboard-to-atmosphere pressure constant	$7\ mol \cdot Pa^{-1} \cdot s^{-1}$	Logar et al. (2012a, 2012b)
k_{PE}	Freeboard extraction pressure constant	$3\ mol \cdot Pa^{-1} \cdot s^{-1}$	Logar et al. (2012a, 2012b)
k_F	Desulphurization rate constant	$1 \cdot 10^{5}\ s^{-1}$	X. Zhang et al. (2019)
k_C	Electrode oxidation rate constant	$1.25 \cdot 10^{4}\ mol \cdot s^{-1}$	Logar et al. (2012a, 2012b)
k_{PBR}	Packed bed reactor flux constant	$10^{-8}\ mol \cdot m^{-1} \cdot Pa^{-1} \cdot s^{-1}$	Assumed value*
k_{Ch}	Channelling flux constant	$2 \cdot 10^{-6}\ mol \cdot m^{-2} \cdot Pa^{-1} \cdot s^{-1}$	Assumed value*

Furnace dimensions			
A	Bath area	$300\ m^2$	Crundwell et al. (2011a)
p_{SAF}	Bath perimeter	$80\ m$	Crundwell et al. (2011a)
V_G	Freeboard volume	$150\ m^3$	Logar et al. (2012a, 2012b)
Heat generation- and transfer constants			
$V_{electrode}$	Electrode voltage	$120\ V$	Crundwell et al. (2011a)
α	Joule heating constant	$3 \cdot 10^{-4}\ V^2 \cdot kW^{-1} \cdot K^{-1}$	Sheng et al. (1998a, 1998b)
R_0	Reference slag resistance	$0.21\ V^2 \cdot kW^{-1}$	Sheng et al. (1998a, 1998b)
$T_{S,\,0}$	Reference slag temperature	$1900\ K$	Eksteen (2011)
$h_{C(S):C(B)}$	Heat transfer coefficient between smelting- and bulk concentrate	$2.75 \cdot 10^{-2}\ kW \cdot m^{-2} \cdot K^{-1}$	Pan et al. (2011)
$h_{G:C(B)}$	Heat transfer coefficient between freeboard and bulk concentrate	$5.5 \cdot 10^{-2}\ kW \cdot m^{-2} \cdot K^{-1}$	Pan et al. (2011)
$h_{S:C(S)}$	Heat transfer coefficient between slag and smelting concentrate	$3.1 \cdot 10^{-1}\ kW \cdot m^{-2} \cdot K^{-1}$	Pan et al. (2011)
$h_{M:S}$	Heat transfer coefficient between liquid matte- and slag	$8.5 \cdot 10^{-2}\ kW \cdot m^{-2} \cdot K^{-1}$	Pan et al. (2011)
$h_{W:S}$	Heat transfer coefficient between cooling units and liquid slag	$7 \cdot 10^{-3}\ kW \cdot m^{-1} \cdot K^{-1}$	Pan et al. (2011)
$h_{W:M}$	Heat transfer coefficient between cooling units and liquid matte	$2.7 \cdot 10^{-2}\ kW \cdot m^{-1} \cdot K^{-1}$	Pan et al. (2011)

Cooling water constants			
F_W	Cooling water flowrate	$2400\ mol \cdot s^{-1}$	Crundwell et al. (2011a)
N_W	Moles of cooling water	$10^4\ mol$	Assumed value*
$T_{W,0}$	Cooling water inlet temperature	$300\ K$	Assumed value*
Other constants			
R	Universal gas constant	$8.314\ J \cdot mol^{-1} \cdot K^{-1}$	
g	Gravitational acceleration constant	$9.81\ m \cdot s^{-2}$	
ε_C	Concentrate bed void fraction	0.4	Eksteen (2011)
P_{atm}	Atmospheric pressure	$101325\ Pa$	
P_{ext}	Extraction pressure	$101315\ Pa$	Logar et al. (2012a, 2012b)
T_{charge}	Concentrate charging temperature	$700\ K$	Crundwell et al. (2011a)
T_{melt}	Concentrate melting temperature	$1500\ K$	Crundwell et al. (2011a)
T_{atm}	Atmospheric temperature	$300\ K$	

* Parameter values were assumed when they were unavailable in literature. Values for these parameters were obtained iteratively by comparing the data generated by the furnace model to the steady-state values reported for SAFs in section 4.5.

APPENDIX C – FURNACE MODEL DEGREES OF FREEDOM ANALYSIS

First, a DOF-analysis is performed for the state variables of the ODE model:			
1.	$\frac{dN_{C(B),XO}}{dt} = \boldsymbol{F_{charge,XO}} - \boldsymbol{F_{mix,XO}}$	+2	2
2.	$\frac{dN_{C(B),XS}}{dt} = \boldsymbol{F_{charge,XS}} - \boldsymbol{F_{mix,XS}} + \boldsymbol{r_{F,C(B)}V_{C(B)}}$	+4	6
3.	$\frac{dN_{C(B),XS_2}}{dt} = \boldsymbol{F_{charge,XS_2}} - \boldsymbol{F_{mix,XS_2}} - r_{F,C(B)}V_{C(B)}$	+2	8
4.	$\frac{dT_{C(B)}}{dt} = \frac{\boldsymbol{Q_{C(S):C(B)}}+\boldsymbol{Q_{G:C(B)}}+c_{P,C}(T_{charge}-T_{C(B)})\cdot F_{charge}}{\boldsymbol{N_{C(B)}}\cdot c_{P,C}}$	+3	11
5.	$\frac{dN_{C(S),XO}}{dt} = F_{mix,XO} - \boldsymbol{F_{melt,XO}}$	+1	12
6.	$\frac{dN_{C(S),XS}}{dt} = F_{mix,XS} - \boldsymbol{F_{melt,XS}} + \boldsymbol{r_{F,C(S)}V_{C(S)}}$	+3	15
7.	$\frac{dN_{C(S),XS_2}}{dt} = F_{mix,XS_2} - r_{F,C(S)}V_{C(S)}$	+0	15
8.	$\frac{dT_{C(S)}}{dt} = \frac{\boldsymbol{Q_{S:C(S)}}\left(1-\frac{T_{C(S)}}{T_{C,melt}}\right)-Q_{C(S):C(B)}+(T_{C(B)}-T_{C(S)})(\boldsymbol{F_{mix}}c_{P,C}+r_{F,C(B)}V_{C(B)}c_{P,G})}{\boldsymbol{N_{C(S)}}c_{P,C}}$	+3	18
9.	$\frac{dN_{C(R)}}{dt} = r_{F,C(B)}V_{C(B)} + r_{F,C(S)}V_{C(S)} + \boldsymbol{r_C} - \boldsymbol{J_R}A$	+2	20
10.	$\frac{dN_S}{dt} = F_{melt,XO} - F_{tap,S}$	+0	20
11.	$\frac{dT_S}{dt} = \frac{\boldsymbol{Q_J}+\boldsymbol{Q_{M:S}}+\boldsymbol{Q_{W:S}}-Q_{S:C(S)}+(T_{C,melt}-T_S)(F_{melt,XO}c_{P,S}+F_{melt,XS}c_{P,M})}{N_S c_{P,S}}$	+3	23
12.	$\frac{dN_M}{dt} = F_{melt,XS} - F_{tap,M}$	+0	23
13.	$\frac{dT_M}{dt} = \frac{\boldsymbol{Q_{W:M}}-Q_{M:S}+(T_S-T_M)F_{melt,XS}c_{P,M}}{N_M c_{P,M}}$	+1	24
14.	$\frac{dN_{G,A}}{dt} = \boldsymbol{F_{neg.P}} - \boldsymbol{F_{pos.P,A}} - \boldsymbol{F_{ext,A}}$	+3	27
15.	$\frac{dN_{G,R}}{dt} = J_R A - \boldsymbol{F_{pos.P,R}} - \boldsymbol{F_{ext,R}}$	+2	29
16.	$\frac{dT_G}{dt} = \frac{c_{P,G}[(T_{C(S)}-T_G)J_R A+(T_{atm}-T_G)F_{neg.P}]-Q_{G:C(B)}}{\boldsymbol{N_G}c_{P,G}}$	+1	30
17.	$\frac{dT_W}{dt} = \frac{c_{P,W}(T_{W,0}-T_W)F_W-Q_{W:M}-Q_{W:S}}{N_W c_{P,W}}$	+0	30
The state space of the ODE model has 30 degrees of freedom. The charge rates of components to the bulk concentrate are functions of one dependent variable, reducing the DOF to 28:			
18.	$F_{charge,XO} = \boldsymbol{F_{charge}}x_{charge,XO}$	+0	30
19.	$F_{charge,XS} = F_{charge}x_{charge,XS}$	-1	29

20.	$F_{charge,\,XS_2} = F_{charge} x_{charge,\,XS_2}$	-1	28
Bulk concentrate mixing rates, desulphurization reaction rates and electrode oxidation reaction rates are considered next to reduce the DOF to 25:			
21.	$F_{mix,\,XO} = k_v V_{C(B)} \boldsymbol{C_{C(B),\,XO}}$	+0	28
22.	$F_{mix,\,XS} = k_v V_{C(B)} \boldsymbol{C_{C(B),\,XS}}$	+0	28
23.	$F_{mix,\,XS_2} = k_v V_{C(B)} \boldsymbol{C_{C(B),\,XS_2}}$	+0	28
24.	$r_{F,\,C(B)} = k_F e^{\left(-\frac{E_{A,\,F}}{RT_{C(B)}}\right)} C_{C(B),\,XS_2}$	-1	27
25.	$r_{F,\,C(S)} = k_F e^{\left(-\frac{E_{A,\,F}}{RT_j}\right)} C_{C(S),\,XS_2}$	-1	26
26.	$r_C = k_C e^{\left(-\frac{E_{A,\,C}}{RT_S}\right)}$	-1	25
Smelting reaction equations further reduce the DOF to 23:			
27.	$F_{melt,\,XO} = \frac{Q_{S:C(S)}\left(\frac{T_{C(S)}}{T_{C,\,melt}}\right)}{\lambda_C + c_{P,\,C}\left(T_{C,\,melt} - T_{C(S)}\right)} \cdot \frac{N_{C(S),\,XO}}{N_{C(S),\,XO} + N_{C(S),\,XS}}$	-1	24
28.	$F_{melt,\,XS} = \frac{Q_{S:C(S)}\left(\frac{T_{C(S)}}{T_{C,\,melt}}\right)}{\lambda_C + c_{P,\,C}\left(T_{C,\,melt} - T_{C(S)}\right)} \cdot \frac{N_{C(S),\,XS}}{N_{C(S),\,XO} + N_{C(S),\,XS}}$	-1	23
Molar flows in- and out of the freeboard are considered next:			
29.	$F_{neg.\,P} = \begin{cases} k_{PR}(P_{atm} - \boldsymbol{P_G}) \; if \; P_{atm} \geq P_G \\ 0 \; if \; P_{atm} < P_G \end{cases}$	+0	23
30.	$F_{pos.\,P,\,R} = \begin{cases} \frac{k_{PR} N_{G,\,R}(P_G - P_{atm})}{N_{G,\,A} + N_{G,\,R}} \; if \; P_G \geq P_{atm} \\ 0 \; if \; P_G < P_{atm} \end{cases}$	-1	22
31.	$F_{pos.\,P,\,A} = \begin{cases} \frac{k_{PR} N_{G,\,A}(P_G - P_{atm})}{N_{G,\,A} + N_{G,\,R}} \; if \; P_G \geq P_{atm} \\ 0 \; if \; P_G < P_{atm} \end{cases}$	-1	21
32.	$F_{ext,\,R} = \frac{k_{PE} N_{G,\,R}(P_G - P_{ext})}{N_{G,\,A} + N_{G,\,R}}$	-1	20
33.	$F_{ext,\,A} = \frac{k_{PE} N_{G,\,A}(P_G - P_{ext})}{N_{G,\,A} + N_{G,\,R}}$	-1	19
Heat generation- and transfer equations bring the DOF down to 13:			
34.	$Q_J = \frac{V_{electrodes}^2}{R_0\left(1 + \alpha\left[T_S - T_{S,\,0}\right]\right)}$	-1	18
35.	$Q_{C(S):C(B)} = h_{C(S):C(B)} A_{SAF}\left(T_{C(S)} - T_{C(B)}\right)$	-1	17
36.	$Q_{G:C(B)} = h_{G:C(B)} A_{SAF}\left(T_G - T_{C(B)}\right)$	-1	16

37.	$Q_{S:C(S)} = h_{S:C(S)}A_{SAF}\left(T_S - T_{C(S)}\right)$	-1	15
38.	$Q_{M:S} = h_{M:S}A_{SAF}(T_M - T_S)$	-1	14
39.	$Q_{W:S} = h_{W:S}p_{SAF}\boldsymbol{L_S}(T_W - T_S)$	+0	14
40.	$Q_{W:M} = h_{W:M}p_{SAF}\boldsymbol{L_M}(T_W - T_M)$	+0	14
Finally, the expressions used to emulate blowbacks in the ODE model are considered next:			
41.	$J_R = \begin{cases} \boldsymbol{J_{R,\,PBR}}\ if\ \boldsymbol{\Delta P_{C(R):G}} \leq c\boldsymbol{\Delta P_{crit}} \\ \boldsymbol{J_{R,\,Ch}}\ if\ \Delta P_{C(R):G} > c\Delta P_{crit} \end{cases} if\ J_R = J_{R,\,PBR}$ $\begin{cases} J_{R,\,PBR}\ if\ \Delta P_{C(R):G} \leq \Delta P_{crit} \\ J_{R,\,Ch}\ if\ \Delta P_{C(R):G} > \Delta P_{crit} \end{cases} if\ J_R = J_{R,\,Ch}$	+3	17
42.	$J_{R,\,PBR} = \frac{k_{PBR}}{\boldsymbol{L_C}}\Delta P_{C(R):G}$	+0	17
43.	$J_{R,\,Ch} = k_{Ch}\Delta P_{C(R):G}$	-1	16
44.	$\Delta P_{crit} = \rho_{C,\,bulk}gL_C$	-1	15
All the equations used to derive the ODE model have been considered. The rest of this DOF analysis presents the equations that relate different variables with each other. First, the moles in the bulk concentrate, smelting concentrate- and freeboard zones are considered:			
45.	$N_{C(B)} = N_{C(B),\,XO} + N_{C(B),\,XS} + N_{C(B),\,XS_2}$	-1	14
46.	$N_{C(S)} = N_{C(S),\,XO} + N_{C(S),\,XS} + N_{C(S),\,XS_2}$	-1	13
47.	$N_G = N_{G,\,A} + N_{G,\,R}$	-1	12
The volumes of bulk- and smelting concentrate appeared in multiple equations, but are functions of the mass of the respective concentrate zones:			
48.	$V_{C(B)} = \frac{M_{XO}N_{C(B),\,XO} + M_{XS}N_{C(B),\,XS} + M_{XS_2}N_{C(B),\,XS_2}}{\rho_{C,\,bulk}}$	-1	11
49.	$V_{C(S)} = \frac{M_{XO}N_{C(S),\,XO} + M_{XS}N_{C(S),\,XS} + M_{XS_2}N_{C(S),\,XS_2}}{\rho_{C,\,bulk}}$	-1	10
The concentrations of materials in the bulk- and smelting concentrate zones appeared in multiple equations, and are functions of other variables:			
50.	$C_{C(B),\,XO} = \frac{N_{C(B),\,XO}}{V_{C(B)}}$	-1	9
51.	$C_{C(B),\,XS} = \frac{N_{C(B),\,XS}}{V_{C(B)}}$	-1	8
52.	$C_{C(B),\,XS_2} = \frac{N_{C(B),\,XS_2}}{V_{C(B)}}$	-1	7
53.	$C_{C(S),\,XS_2} = \frac{N_{C(S),\,XS_2}}{V_{C(S)}}$	-1	6

The matte-, slag- and concentrate bed heights are functions of other variables:			
54.	$L_S = \frac{M_{XO} N_S}{\rho_S A_{SAF}}$	-1	5
55.	$L_M = \frac{M_{XS} N_M}{\rho_M A_{SAF}}$	-1	4
56.	$L_C = \frac{V_{C(B)} + V_{C(S)}}{A_{SAF}}$	-1	3
The concentrate bed height is controlled by the charging rate to the bulk concentrate zone:			
57.	$F_{charge} = \begin{cases} \left.\begin{matrix} F \; if \; L_C < L_{C,\,upper\,lim.} \\ 0 \; if \; L_C \geq L_{C,\,upper\,lim.} \end{matrix}\right\} if \; F_{charge} = F \\ \left.\begin{matrix} F \; if \; L_C < L_{C,\,lower\,lim.} \\ 0 \; if \; L_C \geq L_{C,\,lower\,lim.} \end{matrix}\right\} if \; F_{charge} = 0 \end{cases}$	-1	2
Finally, the pressures in both the freeboard- and concentrate bed voids are determined by the ideal gas law:			
58.	$P_G = \frac{N_G R T_G}{V_G}$	-1	1
59.	$\Delta P_{C(R):G} = \boldsymbol{P_{C(R)}} - P_G$	+0	1
60.	$P_{C(R)} = \frac{N_{C(R)} R T_{C(S)}}{\varepsilon_C \boldsymbol{V_C}}$	+0	1
61.	$V_C = V_{C(B)} + V_{C(S)}$	-1	0

APPENDIX D – MATLAB IMPLEMENTATION OF THE DEVELOPED FURNACE MODEL

This chapter expands on the MATLAB implementation of the furnace model developed in chapter 4. Section D.1 presents the main MATLAB script, section D.2 presents the sub-functions that together make up the set of ordinary differential equations solved by the ODE solver. Section D.3 presents sub-functions used to initialize the ODE solver, load furnace model parameters, or stop the ODE solver when differential equations are changed. Finally, section D.4 presents sub-functions used to convert between different representations of variables.

D.1 Main MATLAB script

This section sets up the model. The simulation duration is specified, model parameters are loaded, and the event function used to change governing ODE equations is set:

```
tDuration = 84; % Simulation duration in days.
tStart    = 0; % Simulation starts at zero
tEnd      = tDuration*24*3600; % Simulation ends at 'tEnd',
                               % units in seconds.
tScope    = [tStart, tEnd]; % Simulation time scope; simulation starts
                            % at 'tStart' and ends at 'tEnd'.
p         = parameters; % Load model parameters and store in 'p'
options = odeset('Events',@(t,y) eventFcn(t,y,p),...
       'AbsTol',1e-3,'RelTol',1e-4);
```

ODE variables are initialized, and memory is pre-allocated for data generated by the ODE solver:

```
[N,T] = variableInitialization; % State variable initial values are stored
                                % in 'N' and 'T'. 'N' stores molar amounts,
                                % 'T' stores temperatures.
xi = state2ode(N,T); % The structures that store initial state variable
                     % values are converted to a columnvector 'xi'.

m = size(xi,1); n = 1e7; % 'm' and 'n' are the dimensions of pre-allocated
                         % memory.

mode.c  = 1; % 'mode' is a structure controlling if concentrate is charged,
             % matte/slag is tapped or if the concentrate bed is ruptured.
             % The model initializes with concentrate being charged.
mode.m  = 0; % The 'm' substructure controls if matte is tapped.
             % '1' indicates matte is being tapped.
mode.s  = 0; % The 's' substructure controls if slag is tapped.
             % '1' indicates slag is being tapped.
mode.bb = 0; % The 'bb' substructure controls if the concentrate bed is
             % ruptured. '1' indicates that the bed is ruptured.

odeVariables = zeros(n,m); % 'odeVariables' is a matrix wherein the ODE
                           % solution is stored.
tSimulated = zeros(n,1); % 'tSimulated' is a columnvector where the
                         % time from the ODE solution is stored.
chargingIdx = ones(n,1)*2; % 'chargingIdx' is a columnvector indicating
                           % where concentrate was charged in the ODE
                           % solution. 'chargingIdx' is used to calculate
                           % the throughput of the modelled SAF.
```

```
bbIdx = ones(n,1)*2;
eventIdx = zeros(n,1); % 'eventIdx' is a columnvector that keeps track of
                       % which events triggered 'eventFcn'.

j = 1; % 'j' is crucial to this model. It keeps track of where solutions
       % from the ODE model should be stored in the pre-allocated memory.

odeVariables(j,:) = xi; % The initial values in 'xi' is stored in the first
                        % row of 'odeVariables'.
chargingIdx(j) = mode.c; % The first entry in 'chargingIdx' is the value
                         % stored in 'mode.c'.
```

The ODE is solved and 'odeVariables' and 'tSimulated' are updated with the solutions. The ODE function is stopped when the event function is triggered. The 'mode' hyperparameter is updated according to which event was triggered, and the ODE function resumes. When 'tEnd' is reached, the simulation stops.

```
while tStart<tEnd
% 'ode15s' solves the SAF model within the span specified by 'tScope' using
% the initial values in 'xi'. The 'furnaceModelODEs' subfunction is used in
% the 'ode15s' solver. The output generated by 'ode15s' is in 'yout'. 'ie'
% is the number of the specific event triggered, as given in the 'eventFcn'
% subfunction.
[tout,yout,~,~,ie] = ode15s(@(t,x) furnaceModelODEs(t,x,p,mode),tScope,...
                            xi,options);
% 'ode15s' stops if 'eventFcn' is triggered or 'tEnd' is reached. If 'tEnd'
% is reached, ie will be empty. '0' is assigned to ie to allow 'switch' to
% work.
if isempty(ie) == 1
    ie = 0;
end
% 'tStart', 'xi' and 'tScope' are updated.
tStart = tout(end);
tScope = [tStart tEnd];
xi = yout(end,:)';
% The values in 'yout' and 'tout' are assigned to 'odeVariables' and
% 'tSimulated'. The indexing variable, j, is updated.
nSample = size(yout,1)-2;
odeVariables(j+1:j+nSample,:) = yout(2:end-1,:);
tSimulated(j+1:j+nSample,:) = tout(2:end-1,:);
chargingIdx(j+1:j+nSample) = mode.c;
bbIdx(j+1:j+nSample) = mode.bb;
eventIdx(j+1:j+nSample) = ie;
j = j+nSample;
% If two events specified in the event funtions are triggered
% simultaneously, then ie is a rowvector with entries specifying which
% events were triggered. A for loop cycles through these entries.
ieSize = size(ie,2);
for i = 1:ieSize
    g = ie(i);
    switch g
     case 1
        mode.c = 1;
     case 2
        mode.c = 0;
     case 3
```

```
            mode.s = 0;
        case 4
            mode.s = 1;
        case 5
            mode.m = 0;
        case 6
            mode.m = 1;
        case 7
            mode.bb = 1;
        case 8
            mode.bb = 0;
    end
end
end
```

This section simply removes the unused pre-allocated memory. Furthermore, the first hour's data generated by the model is removed.

```
idxDelete = [(j+1):n]';
odeVariables(idxDelete,:) = [];
chargingIdx(idxDelete) = [];
bbIdx(idxDelete) = [];
tSimulated(idxDelete) = [];
idxDelete = [1:find(tSimulated>3600,1)]';
odeVariables(idxDelete,:) = [];
chargingIdx(idxDelete) = [];
bbIdx(idxDelete) = [];
tSimulated(idxDelete) = [];
```

The values in 'odeVariables' correspond to variable steps in 'tSimulated'. This section creates a new dataset, 'odeData', with entries obtained through interpolation from 'odeVariables', corresponding to fixed time samples in 'tInterp':

```
tInterp = [3600:10:tEnd]'; % The simulated process data is sampled every 10
                           % seconds.
tInterp(tInterp<min(tSimulated)) = []; % This ensures that 'tInterp' only
                                       % contains entries at which
                                       % 'tSimulated' can be interpolated.
n = size(tInterp,1);  % 'n' is the number of samples in 'odeData'.
odeData = zeros(n,m); % Pre-allocates memory for 'odeData'.

% 'odeData' is populated through interpolation with 'interp1', using
% 'tSimulated' and 'odeVariables'. 'nearest' interpolation is used:
for i = 1:m
    odeData(:,i) = interp1(tSimulated,odeVariables(:,i),tInterp,'nearest');
end
tInterp = tInterp - min(tInterp); % 'tInterp' starts at zero.
dataProfile = ode2profile(odeData);
t = tInterp; t = t/(24*3600);
simulationData.t = t;
simulationData.data = dataProfile;
```

D.2 ODE sub-functions

Sub-functions directly related to the ODE model are defined here. 'furnaceModelODEs' compute the rate of change in state variables from the state variables in 'x'.

```
function dx = furnaceModelODEs(t,x,p,mode)
% The columnvector, 'x', of state variables are stored in the structures
% 'N' and 'T'. Molar amounts are stored in 'N', temperatures are stored in
% 'T'.
[N,T] = ode2state(x);
[N,V,C,L,P] = state2derived(N,T,p);
% Heat generation and transfer expressions are contained in the 'Q'
% structure.
Q = heatGenerationTransfer(T,L,p);
% Mass transfer expressions are contained in the 'F' structure. The
% concentrate charging rate depends only on the current simulation time,
% 't', and the mode. Subsequent mass transfer expressions are updated with
% the previous expressions:
F = concentrateCharging(mode,t);   % Computes the concentrate charging rate.
F = matteSlagTapping(mode,F);      % Computes the matte/slag tapping rate.
F = concentrateMixing(V,C,p,F);    % Computes the rate of bulk- and
                                   % smelting concentrate mixing.
F = concentrateMelting(Q,T,N,p,F); % Computes the concentrate melting rate.
F = gasFlow(P,N,p,F,t);       % Computes the rate at which gas is exchanged
                              % between the freeboard and atmosphere.
J = gasFlux(P,L,mode,p);    % Computes the reaction gas flux rate through
                            % the concentrate bed.
r = reactionRates(T,C,p);  % Computes the rate at which reaction gases are
                           % formed.
dN = molarChange(F,r,J,V,p); % Computes the rate of change in molar state
                             % variables.
dT = temperatureChange(T,Q,N,F,r,J,V,p,t); % Computes the rate of change in
                                           % state temperature variables.
dx = state2ode(dN,dT); % Converts the rate of change in the 'dN' and 'dT'
                       % expressions to a columnvector, 'dx'.
end
```

'concentrateCharging' computes the total concentrate charging rate, as well as the rate at which slag, matte and sulphurized matte components are charged.

```
function F = concentrateCharging(mode,t)
F.charge.total = 410*mode.c; % Total concentrate charging rate, mol/s.

s = sin(t*pi/(7*24*3600))>=0; % Every week, the composition of feed shifts
                              % from Merensky (M) ore to Merensky-UG2 (MU)
                              % ore blend. 's' is used to switch between
                              % these feed types.
% The molar flowrate of slag, matte and sulphurized components in the feed
% are contained in the 'XO', 'XS' and 'XS2' substructures of 'F.charge'.
F.charge.XO  = F.charge.total*(0.78+0.12*s); % MU has more slag than M
F.charge.XS  = F.charge.total*(0.21-0.12*s); % MU has less matte than M
F.charge.XS2 = F.charge.total*0.01; % Both ores have similar sulphurized
                                    % matte compositions.
end
```

'matteSlagTapping' computes the matte- and slag tapping rates:

```
function F = matteSlagTapping(mode,F)
F.matteTap = 100*mode.m; % Matte tapping rate, mol/s.
F.slagTap = 400*mode.s;  % Slag tapping rate, mol/s.
end
```

'concentrateMixing' computes the rate of transfer from the bulk- to smelting concentrate:

```
function F = concentrateMixing(V,C,p,F)
kv = p.k.V*(V.CB/V.CS); % 'kv' increases if the volume ratio between the
                        % bulk- and smelting concentrate increases. This
                        % avoids numerical instability when the amount of
                        % smelting concentrate reaches zero, with bulk
                        % concentrate remaining.
F.mix.total = kv*V.CB*(C.CB.XO+C.CB.XS+C.CB.XS2); % Total transfer rate, mol/s
F.mix.XO    = kv*V.CB*C.CB.XO; % Transfer rate of slag, mol/s
F.mix.XS    = kv*V.CB*C.CB.XS; % Transfer rate of matte, mol/s
F.mix.XS2   = kv*V.CB*C.CB.XS2; % Transfer rate of sulphurized matte, mol/s
end
```

'concentrateMelting' computes the rate at which slag- and matte components melts from the smelting concentrate.

```
function F = concentrateMelting(Q,T,N,p,F)
F.melt.XO = Q.S2CS*((T.CS/p.other.Tmelt)^2)/(p.fus.C-p.cP.C*(p.other.Tmelt-T.CS))*...
                N.CS.XO/(N.CS.XO+N.CS.XS); % Slag melting rate, mol/s.
F.melt.XS = Q.S2CS*((T.CS/p.other.Tmelt)^2)/(p.fus.C-p.cP.C*(p.other.Tmelt-T.CS))*...
                N.CS.XS/(N.CS.XO+N.CS.XS); % Matte melting rate, mol/s.
end
```

'gasFlow' computes the molar exchange rate between the freeboard and the atmosphere.

```
function F = gasFlow(P,N,p,F,t)
F.negP   = p.k.PR*(p.other.Patm-P.G)...
            *(p.other.Patm>=P.G); % Air drawn in, mol/s
F.posP.R = p.k.PR*(P.G-p.other.Patm)...
            *(P.G>=p.other.Patm)...
            *N.G.R/(N.G.R+N.G.A); % Reaction gases blown out, mol/s
F.posP.A = p.k.PR*(P.G-p.other.Patm)...
            *(P.G>=p.other.Patm)...
            *N.G.A/(N.G.R+N.G.A); % Air blown out, mol/s
s = 2*(sin(pi*t/(2.5*7*24*3600))<0); % Every two and a half weeks, a fault
                                     % is introduced to the extraction fan.
                                     % The extraction pressure increases by
                                     % 2, increasing the freeboard
                                     % pressure.
fExt = p.k.PE*(P.G-(p.other.Pext+s)); % Total rate of freeboard gas
                                      % extraction, mol/s.
F.ext.R  = fExt*N.G.R/(N.G.R+N.G.A); % Reaction gas extraction, mol/s
F.ext.A  = fExt*N.G.A/(N.G.R+N.G.A); % Air extraction, mol/s
end
```

'heatGenerationTransfer' computes the rate of heat generation in the slag zone, as well as the rate of heat transfer between zones. Ohmic heating is assumed to compute the rate of heat generation.

```
function Q = heatGenerationTransfer(T,L,p)
Q.J = (p.Q.Velectrode^2)...
    /(p.Q.R0*(1+p.Q.alpha*(T.S-p.Q.T0))); % Heat generation rate, kW
Q.CS2CB = p.Q.hCS2CB*p.dim.A*(T.CS-T.CB); % Heat transfer from smelting-
                                          % to bulk concentrate, kW
Q.G2CB  = p.Q.hG2CB*p.dim.A*(T.G-T.CB); % Heat transfer from freeboard
                                        % to bulk concentrate, kW
Q.S2CS  = p.Q.hS2CS*p.dim.A*(T.S-T.CS); % Heat transfer from slag to
                                        % smelting concentrate, kW
Q.M2S   = p.Q.hM2S*p.dim.A*(T.M-T.S); % Heat transfer from matte to slag, kW
Q.W2S   = p.Q.hW2S*p.dim.p*L.S*(T.W-T.S); % Heat transfer from cooling units
                                          % to slag, kW
Q.W2M   = p.Q.hW2M*p.dim.p*L.M*(T.W-T.M); % Heat transfer from cooling units
                                          % to matte, kW
end
```

'gasFlux' computes the reaction gas flux from the concentrate bed to the furnace freeboard.

```
function J = gasFlux(P,L,mode,p)
J_pbr = (P.CR-P.G)*p.k.PBR/L.C; % Flux when concentrate bed is unruptured
                                % and behaves as a PBR, mol/m^2.s
J_ch  = (P.CR-P.G)*p.k.Ch; % Flux when concentrate bed is ruptured and
                           % channels form, mol/m^2.s
J = J_pbr*(mode.bb==0)+J_ch*(mode.bb==1); % The output of this function
                                          % depends on whether or not the
                                          % concentrate bed is ruptured.
J = J*(J>=0); % 'J' is either positive or zero, this prevents numerical
              % instability at the start of the furnace simulation.
end
```

'reactionRates' computes the desulphurization reaction rates in the bulk- and smelting concentrate zones, as well as the rate of electrode oxidation.

```
function r = reactionRates(T,C,p)
r.F.CB = p.k.rF*exp(-p.EA.F/(p.other.R*T.CB))...
            *C.CB.XS2; % Desulphurization in bulk concentrate, mol/s
r.F.CS = p.k.rF*exp(-p.EA.F/(p.other.R*T.CS))...
            *C.CS.XS2; % Desulphurization in smelting concentrate, mol/s
r.C    = p.k.rC*exp(-p.EA.C/(p.other.R*T.S)); % Electrode oxidation, mol/s
end
```

The molar transfer- and reaction rates are compiled in 'molarChange' to compute the rate of change in state variable molar amounts.

```
function dN = molarChange(F,r,J,V,p)
dN.CB.XO  = F.charge.XO-F.mix.XO; % Bulk concentrate slag, mol/s
dN.CB.XS  = F.charge.XS-F.mix.XS...
               +r.F.CB*V.CB; % Bulk concentrate matte, mol/s
```

```
dN.CB.XS2 = F.charge.XS2-F.mix.XS2...
                -r.F.CB*V.CB; % Bulk concentrate sulphurized matte, mol/s
dN.CS.XO  = F.mix.XO-F.melt.XO;    % Smelting concentrate slag, mol/s
dN.CS.XS  = F.mix.XS-F.melt.XS...
                +r.F.CS*V.CS;      % Smelting concentrate matte, mol/s
dN.CS.XS2 = F.mix.XS2-r.F.CS*V.CS; % Smelting concentrate sulphurized matte, mol/s
dN.CR = r.F.CB*V.CB+r.F.CS*V.CS...
                +r.C-J*p.dim.A; % Reaction gases in the concentrate bed, mol/s
dN.S  = F.melt.XO-F.slagTap; % Change in slag zone, mol/s
dN.M  = F.melt.XS-F.matteTap; % Change in matte zone, mol/s
dN.G.A = F.negP - F.posP.A - F.ext.A; % Change in air in freeboard, mol/s
dN.G.R = J*p.dim.A - F.posP.R - F.ext.R; % Change in reaction gases in freeboard, mol/s
end
```

'temperatureChange' compiles mass- and heat transfer expressions to compute the rate of change in state temperature variables.

```
function dT = temperatureChange(T,Q,N,F,r,J,V,p,t)
% Bulk concentrate temperature change, K/s:
dT.CB = (Q.CS2CB+Q.G2CB+p.cP.C*(p.other.Tcharge-T.CB)*F.charge.total)/...
            (N.CB.total*p.cP.C);
% Smelting concentrate temperature change, K/s:
dT.CS = (Q.S2CS*(1-(T.CS/p.other.Tmelt)^2)-Q.CS2CB+(T.CB-T.CS)*...
            (F.mix.total*p.cP.C+r.F.CB*V.CB*p.cP.G))/(N.CS.total*p.cP.C);
% Slag temperature change, K/s:
dT.S  = (Q.J+Q.M2S+Q.W2S-Q.S2CS+(p.other.Tmelt-T.S)*...
            (F.melt.XO*p.cP.S+F.melt.XS*p.cP.M))/(N.S*p.cP.S);
% Matte temperature change, K/s:
dT.M  = (Q.W2M-Q.M2S+(T.S-T.M)*F.melt.XS*p.cP.M)/(N.M*p.cP.M);
% Freeboard temperature change, K/s:
dT.G  = (p.cP.G*((T.CS-T.G)*J*p.dim.A+(p.other.Tatm-T.G)*F.negP)-Q.G2CB)/...
            (N.G.total*p.cP.G);

s = 0.5 + 0.5*(sin(pi*t/(3*7*24*3600))>=0); % The cooling water flowrate is
                                            % halved every three weeks in
                                            % this simulation.
% Cooling units temperature change, K/s:
dT.W  = (p.cP.W*(p.water.T0-T.W)*p.water.F*s-Q.W2M-Q.W2S)/(p.water.N*p.cP.W);
end
```

D.3 Auxiliary sub-functions

'eventFcn' is used to halt the ode solver when concentrate charging and matte/slag tapping should be started or stopped. The necessary variables are calculated from the ODE variables stored in y:

```
function [threshold,isterminal,direction] = eventFcn(t,y,p)
[N,T] = ode2state(y);
[~,~,~,L,P] = state2derived(N,T,p);
delPcrit = p.D.C*p.other.g*L.C;
delP = P.CR - P.G;
s = sin(t*2*pi/(4*24*3600))<0;
% The ode solver stops when the height of the concentrate bed exceeds its
% thresholds:
threshold(1,1) = L.C - (0.4 + 0.3*s);
```

```
threshold(2,1) = L.C - (0.6 + 0.5*s);
% The ode solver also stops when the slag height exceeds its thresholds:
threshold(3,1) = L.S - 0.7; % Height of slag goes below 0.1 m
threshold(4,1) = L.S - 1.1; % Height of slag goes above 0.2 m
% The ode solver also stops when the matte height exceeds its thresholds:
threshold(5,1) = L.M - 0.3; % Height of matte goes below 0.04 m
threshold(6,1) = L.M - 0.5; % Height of matte goes above 0.08 m
% The ode solver stops when the pressure difference over the concentrate bed
% exceeds its thresholds:
threshold(7,1) = delP - delPcrit*4; % Goes above pcrit
threshold(8,1) = delP - delPcrit*1; % Goes below pcrit
% If isterminal(i) = 1 when threshold(i) crosses 0, then the ode solver
% stops. This happens for each threshold.
isterminal(1,1) = 1;
isterminal(2,1) = 1;
isterminal(3,1) = 1;
isterminal(4,1) = 1;
isterminal(5,1) = 1;
isterminal(6,1) = 1;
isterminal(7,1) = 1;
isterminal(8,1) = 1;

% direction(i) determines if threshold(i) should be crossed by decreasing
% or increasing values for the ode solver to be stopped:
direction(1,1) = -1;
direction(2,1) = 1;
direction(3,1) = -1;
direction(4,1) = 1;
direction(5,1) = -1;
direction(6,1) = 1;
direction(7,1) = 1;
direction(8,1) = -1;
end
```

'variableInitialization' stores the initial state variable values in 'N' and 'T'.

```
function [N,T] = variableInitialization
T.CB = 1100; % Initial bulk concentrate temperature, K
T.CS = 1400; % Initial smelting concentrate temperature, K
T.S  = 1900; % Initial slag zone temperature, K
T.M  = 1750; % Initial matte zone temperature, K
T.G  = 900;  % Initial freeboard temperature, K
T.W  = 300;  % Initial cooling units temperature, K

N.CB.XO  = 7.8e5;  % Initial slag in bulk concentrate, mol
N.CB.XS  = 5.75e5; % Initial matte in bulk concentrate, mol
N.CB.XS2 = 0;      % Initial sulphurized matte in bulk concentrate, mol
N.CS.XO  = 7.8e5;  % Initial slag in smelting concentrate, mol
N.CS.XS  = 5.75e5; % Initial matte in smelting concentrate, mol
N.CS.XS2 = 0;      % Initial sulphurized matte in smelting concentrate, mol
N.CR     = 0; % Initial reaction gas in concentrate bed, mol
N.S      = 1.1e7; % Initial liquid slag, mol
N.M      = 6.5e6; % Initial liquid matte, mol
N.G.A    = 1830;  % Initial air in freeboard, mol
N.G.R    = 0;     % Initial reaction gas in freeboard, mol
end
```

'parameters' loads model parameters and stores them in 'p':

```
function p = parameters
% Molar masses, kg/mol
p.M.G   =  30e-3; % Reaction gases/air
p.M.XO  =  72e-3; % Lumped slag components
p.M.XS  =  88e-3; % Lumped matte components
p.M.XS2 = 120e-3; % Lumped sulfurized components

% Densities, kg/m^3
p.D.C = 1600; % Concentrate density
p.D.S = 2960; % Slag density
p.D.M = 4800; % Matte density

% Heat of fusion, kJ/mol
p.fus.C = 133; % Concentrate heat of fusion
% Heat capacity, kJ/mol.K
p.cP.C =  75e-3; % Concentrate heat capacity
p.cP.S =  99e-3; % Slag heat capacity
p.cP.M =  78e-3; % Matte heat capacity
p.cP.G =  30e-3; % Reaction gases/air heat capacity
p.cP.W =  75e-3; % Cooling water heat capacity
% Activation energy, J/mol
p.EA.F =  150e3; % Desulphurization reaction
p.EA.C =  120e3; % Electrode oxidation reaction
% Rate-determining constants
p.k.V   = 2e-4;   % Concentrate mixing constant
p.k.PR  = 7;      % Freeboard- to atmosphere flow constant, mol/Pa.s
p.k.PE  = 3;      % Freeboard extraction pressure flow constant, mol/Pa.s
p.k.rF  = 1.0e5;  % Desulphurization rate constant, 1/s
p.k.rC  = 1.25e4; % Electrode oxidation rate constant, mol/s
p.k.PBR = 1.0e-8; % PBR flux constant, mol/m.Pa.s
p.k.Ch  = 2e-6;   % Channeling flux constant, mol/m^2.Pa.s
% Furnace dimensions
p.dim.A = 300; % Bath area, m^2
p.dim.p = 80;  % SAF perimeter, m
p.dim.V = 150; % Freeboard volume, m^3
% Heat generation & transfer constants
p.Q.Velectrode = 120;     % Electrode voltage, V
p.Q.alpha      = 0.0003;  % Joule heating constant, V^2/kW.K
p.Q.R0         = 0.21;    % Slag resistivity at 1900 K, V^2/kW
p.Q.T0         = 1900;    % Base slag temperature, K
p.Q.hCS2CB     = 0.0275;  % Heat transfer coefficient between smelting- and
                          % bulk concentrate, kW/m^2.K
p.Q.hG2CB      = 0.055;   % Heat transfer coefficient between freeboard and
                          % bulk concentrate, kW/m^2.K
p.Q.hS2CS      = 0.31;    % Heat transfer coefficient between slag and
                          % smelting concentrate, kW/m^2.K
p.Q.hM2S       = 0.085;   % Heat transfer coefficient between matte and
                          % slag, kW/m^2.K
p.Q.hW2S       = 0.007;   % Heat transfer coefficient between cooling units
                          % and slag, kW/m.K
p.Q.hW2M       = 0.027;   % Heat transfer coefficient between cooling units
                          % and matte, kW/m.K
% Other constants
p.other.R = 8.314; % Universal gas constant, J/mol.K
p.other.g = 9.81;  % Gravity acceleration, m/s^2
```

```
p.other.e = 0.4;   % Concentrate bed voidage
p.other.Patm = 101325; % Atmospheric pressure, Pa
p.other.Pext = 101315; % Extraction pressure, Pa
p.other.Tcharge = 700; % Concentrate charge temperature, K
p.other.Tmelt = 1500;  % Concentrate melting temperature, K
p.other.Tatm  = 300;   % Atmosphere temperature, K
% Cooling water constants
p.water.F  = 2400;  % Cooling water flowrate, mol/s
p.water.N  = 10000; % Cooling water molar amount, mol
p.water.T0 = 300;   % Cooling water temperature, K
end
```

D.4 Representation sub-functions

These functions are used to convert between different representations of the model state variables. 'ode2state' is used to convert from a columnvector of state variables to structured variables of molar amounts and temperatures. 'state2derived' converts from structured state variables to process variables that can be derived from state variables. 'state2ode' converts from the structured representation of state variables to a columnvector of state variables used by the ode solver. 'ode2profile' converts a matrix of ODE data generated by 'ode15s' to a structure of process variables.

```
function [N,T] = ode2state(x)
N.CB.XO  = x(1);  % Moles of slag in the bulk concentrate
N.CB.XS  = x(2);  % Moles of matte in the bulk concentrate
N.CB.XS2 = x(3);  % Moles of sulphurized matte in the bulk concentrate
T.CB     = x(4);  % Bulk concentrate temperature, K
N.CS.XO  = x(5);  % Moles of slag in the smelting concentrate
N.CS.XS  = x(6);  % Moles of matte in the smelting concentrate
N.CS.XS2 = x(7);  % Moles of sulphurized matte in the smelting concentrate
T.CS     = x(8);  % Smelting concentrate temperature, K
N.CR     = x(9);  % Moles of reaction gas trapped in the concentrate
N.S      = x(10); % Moles of liquid slag
T.S      = x(11); % Liquid slag temperature, K
N.M      = x(12); % Moles of liquid matte
T.M      = x(13); % Liquid matte temperature, K
N.G.A    = x(14); % Moles of air in the furnace freeboard
N.G.R    = x(15); % Moles of reaction gas in the furnace freeboard
T.G      = x(16); % Freeboard gas temperature, K
T.W      = x(17); % Cooling water temperature, K
end

function [N,V,C,L,P] = state2derived(N,T,p)
N.CB.total = N.CB.XO + N.CB.XS + N.CB.XS2; % Total moles in bulk concentrate
N.CS.total = N.CS.XO + N.CS.XS + N.CS.XS2; % Total moles in smelting concentrate
N.G.total  = N.G.A + N.G.R; % Total moles in freeboard gas

V.CB = (p.M.XO*N.CB.XO+p.M.XS*N.CB.XS+p.M.XS2*N.CB.XS2)/p.D.C; % Bulk conc. volume, m^3
V.CS = (p.M.XO*N.CS.XO+p.M.XS*N.CS.XS+p.M.XS2*N.CS.XS2)/p.D.C; % Smelting conc. volume, m^3
V.C = V.CB+V.CS; % Total concentrate volume, m^3

C.CB.XO  = N.CB.XO/V.CB; % Slag concentration in bulk concentrate, mol.m^-3
C.CB.XS  = N.CB.XS/V.CB; % Matte concentration in bulk concentrate, mol.m^-3
C.CB.XS2 = N.CB.XS2/V.CB; % Sulphurized matte concentration in bulk concentrate, mol.m^-3
C.CS.XS2 = N.CS.XS2/V.CS; % Sulphurized matte concentration in smelting concentrate, mol.m^-3
```

```
L.S = p.M.XO*N.S/(p.D.S*p.dim.A); % Liquid slag level, m
L.M = p.M.XS*N.M/(p.D.M*p.dim.A); % Liquid matte level, m
L.C = (V.CB+V.CS)/p.dim.A; % Concentrate bed thickness, m

P.G  = N.G.total*p.other.R*T.G/p.dim.V; % Freeboard absolute pressure
P.CR = N.CR*p.other.R*T.CS/(p.other.e*V.C); % Reaction gas in conc. absolute pressure
end

function dx = state2ode(dN,dT)
dx(1,1) = dN.CB.XO;
dx(2,1) = dN.CB.XS;
dx(3,1) = dN.CB.XS2;
dx(4,1) = dT.CB;
dx(5,1) = dN.CS.XO;
dx(6,1) = dN.CS.XS;
dx(7,1) = dN.CS.XS2;
dx(8,1) = dT.CS;
dx(9,1) = dN.CR;
dx(10,1) = dN.S;
dx(11,1) = dT.S;
dx(12,1) = dN.M;
dx(13,1) = dT.M;
dx(14,1) = dN.G.A;
dx(15,1) = dN.G.R;
dx(16,1) = dT.G;
dx(17,1) = dT.W;
end

function profileVariables = ode2profile(odeVariables)
g = odeVariables;
f.N.CB.XO  = g(:,1); % Slag in bulk concentrate, mol
f.N.CB.XS  = g(:,2); % Matte in bulk concentrate, mol
f.N.CB.XS2 = g(:,3); % Sulphurized matte in bulk concentrate, mol
f.T.CB     = g(:,4); % Bulk concentrate temperature, K
f.N.CS.XO  = g(:,5); % Slag in smelting concentrate, mol
f.N.CS.XS  = g(:,6); % Matte in smelting concentrate, mol
f.N.CS.XS2 = g(:,7); % Sulphurized matte in smelting concentrate, mol
f.T.CS     = g(:,8); % Smelting concentrate temperature, K
f.N.CR     = g(:,9); % Reaction gases in concentrate, mol
f.N.S      = g(:,10); % Liquid slag, mol
f.T.S      = g(:,11); % Liquid slag temperature, K
f.N.M      = g(:,12); % Liquid matte, mol
f.T.M      = g(:,13); % Liquid matte temperature, K
f.N.G.A    = g(:,14); % Air in freeboard, mol
f.N.G.R    = g(:,15); % Reaction gases in freeboard, mol
f.T.G      = g(:,16); % Freeboard gas temperature, K
f.T.W      = g(:,17); % Cooling units temperature, K

p = parameters;
f.N.CB.total = f.N.CB.XO + f.N.CB.XS + f.N.CB.XS2; % Total bulk concentrate, mol
f.N.CS.total = f.N.CS.XO + f.N.CS.XS + f.N.CS.XS2; % Total smelting concentrate, mol
f.N.G.total  = f.N.G.A + f.N.G.R; % Total freeboard gas, mol
% Bulk concentrate volume, m^3:
f.V.CB = (p.M.XO*f.N.CB.XO+p.M.XS*f.N.CB.XS+p.M.XS2*f.N.CB.XS2)/p.D.C;
% Smelting concentrate volume, m^3:
f.V.CS = (p.M.XO*f.N.CS.XO+p.M.XS*f.N.CS.XS+p.M.XS2*f.N.CS.XS2)/p.D.C;
f.V.C = f.V.CB+f.V.CS; % Total concentrate volume, m^3
```

```
f.C.CB.XO  = f.N.CB.XO./f.V.CB;  % Bulk concentrate slag concentration, mol/m^3
f.C.CB.XS  = f.N.CB.XS./f.V.CB;  % Bulk concentrate matte concentration, mol/m^3
f.C.CB.XS2 = f.N.CB.XS2./f.V.CB; % Bulk concentrate sulphurized matte concentration, mol/m^3
f.C.CS.XS2 = f.N.CS.XS2./f.V.CS; % Smelting concentrate sulphurized matte concentration, mol/m^3

f.L.S = p.M.XO*f.N.S/(p.D.S*p.dim.A); % Slag liquid level, m
f.L.M = p.M.XS*f.N.M/(p.D.M*p.dim.A); % Matte liquid level, m
f.L.C = (f.V.CB+f.V.CS)/p.dim.A;      % Concentrate bed thickness, m

f.P.G  = f.N.G.total*p.other.R.*f.T.G/p.dim.V; % Freeboard pressure, Pa
f.P.CR = f.N.CR*p.other.R.*f.T.CS...
              ./(p.other.e.*f.V.C); % Reaction gas in concentrate pressure, Pa

f.Q.S2CS  = p.Q.hS2CS*p.dim.A*(f.T.S-f.T.CS); % Heat transfer from slag to concentrate, kW
% Heat generated in slag zone, kW:
f.Q.J     = (p.Q.Velectrode^2)./(p.Q.R0*(1+p.Q.alpha*(f.T.S-p.Q.T0)));
% Concentrate smelting rate, mol/s:
f.F.melt.total = f.Q.S2CS.*(f.T.CS/p.other.Tmelt)./(p.fus.C-p.cP.C*(p.other.Tmelt-f.T.CS));
% Electrode oxidation rate, mol/s:
f.r.C  = p.k.rC*exp(-p.EA.C./(p.other.R*f.T.S));

profileVariables = f;
end
```

APPENDIX E – MATLAB IMPLEMENTATION OF FPR MODELS

All of the fault pattern recognition models evaluated in this thesis were implemented on MATLAB. This appendix chapter provides the MATLAB code used to implement algorithms and equations presented in chapter 5, in the same order that those algorithms appear. First, section E.1 shows how feature engineering techniques are applied. Section E.2 shows how data is partitioned into training, validation and testing datasets. Section E.2 also shows indices for faulty-, normal and target observations are obtained. Section E.3 shows how discriminants are evaluated in a moving window. Section E.4 shows how performance metrics are calculated. Finally, sections E.5 to E.8 show how model parameters are derived from data, given specified design parameters, as well as how reconstruction errors are computed. Each function presented in this chapter is preceded by a brief description of that function, and is succeeded by a table describing each input to- and output from the function.

E.1 MATLAB implementation of feature engineering

The function used to apply the feature engineering techniques given in section 5.1 is given below:

```
function Xe = applyFeatureEngineering(X,fe)
    if fe==1 % Feature engineering is applied if fe = 1.
        X(:,1) = X(:,1)-movmean(X(:,1),[1560,0]); % Liquid slag level.
        X(:,2) = X(:,2)-movmean(X(:,2),[2490,0]); % Liquid matte level.
        X(:,3) = X(:,3)-movmean(X(:,3),[2100,0]); % Slag temperature
        % Matte temperature:
        X(:,4) = movmean(X(:,4)-movmean(X(:,4),[2700,0]),[570,0]);
        % Bulk concentrate temperature:
        X(:,5) = X(:,5)-movmean(X(:,5),[1020,0]);
        % Freeboard temperature:
        X(:,6) = movstd(X(:,6)-movmean(X(:,6),[960,0]),[630,0]);
        % Cooling water temperature:
        X(:,7) = X(:,7)-movmean(X(:,7),[1230,0]);
        % Freeboard reaction gas concentration:
        X(:,8) = movstd(X(:,8) - movmean(X(:,8),[1110,0]),[660,0]);
        % Freeboard gauge pressure:
        X(:,9) = movstd(X(:,9)-movmean(X(:,9),[2160,0]),[570,0]);
    end
    Xe = X;
end
```

Function inputs	
fe	Logical value indicating if feature engineering is applied.
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
Function outputs	
Xe	Matrix of feature engineered variables, different variables are in different columns and different samples are in different rows.

E.2 MATLAB implementation of data partitioning

The function used to define training, validation and testing indices, ‘partitionData’, is given below. Using this function, observations are partitioned as presented in Figure 5.3.

```
function [idxTrain,idxValidation,idxTest] = partitionData(t)
    n = size(t,1); % Total number of samples in dataset
    tVal = t(ceil(n/3)); % First third of data is used for training
    tTest = t(ceil(2*n/3)); % Second third of data is used for validation

    idxTrain = t<tVal; % All observations from first third of simulation
                       % used in training set.
       % All observations from the first two thirds of the simulation, not in
    % the training set, are used in the validation set.
    idxValidation = ((t<tTest).*(~idxTrain))==1;
    % All observations not in the training or validation set are used for
    % testing:
    idxTest  = ((~idxTrain).*(~idxValidation))==1;
end
```

Function input	
t	Column vector of the times where process data is sampled.
Function outputs	
idxTrain	Logical column vector, with ones indicating observations in the training set.
idxValidation	Logical column vector, with ones indicating observations used for validation.
idxTest	Logical column vector, with ones indicating observations used for model testing.

The function used to calculate which observations are faulty and which are normal (as presented in Figure 5.4) is given below. This function is also used to select target observations from the training set (as presented in Figure 5.6):

```
function [idxTarget,idxFault,idxNormal] = findGroundTruth(t,P,idxTrain,...
                                minimumWarning,predictionPeriod)
    % Indices where blowbacks occur are identified where the pressure, 'P',
    % is positive:
    idxPP = P>=0;
    n = size(idxPP,1); % Number of observations in the data
    a = find(idxPP); % Index numbers of blowbacks
    b = [predictionPeriod+1; diff(a)]; % Returns the number of samples
                                       % preceding each blowback since the
                                       % previous blowback was recorded.
    c = a(b>predictionPeriod); % Returns the indices of blowbacks that are
                               % not preceded by other blowbacks within the
                               % number of samples specified by
                               % 'predictionPeriod'.
    d = c-predictionPeriod; % Returns the indices where target windows start.
    e = c-minimumWarning; % Returns indices where target window ends.
    f = zeros(n,1);
    f(d) = 1; f(e) = -1;
    g = cumsum(f)==1; % Returns logical index of target window.
    idxTarget = (idxTrain.*g)==1; % Only indices within the training
                                  % data are used.
    a = find(idxPP); % Indices of blowbacks.
    b = [predictionPeriod+1; diff(a)];
    c = a(b>predictionPeriod); % Indices of blowbacks not preceded by
                               % other blowbacks within the prediction
                               % period.
    d = c-minimumWarning; % Indices where target windows end.
```

```
    e = find(b>predictionPeriod)-1;
    f = [a(e(2:end)); a(end)]+predictionPeriod; % Indices where blowback windows end.
    g = zeros(n,1); g(d) = 1; g(f) = -1;
    h = cumsum(g)==1; % Blowback window indices. Recognizing blowback-
                      % preceding conditions here are not incorrect, but
                      % they do not contribute to operator safety.
       % Faulty observations occur every two days, for two days. Note that the
       % blowback windows themselves are removed.
    idxFault  = ((sin(pi*t/(2*24*3600))<0).*(~h))==1;
    idxNormal = ((~idxFault).*(~h))==1;
end
```

Function inputs	
t	Column vector of the times where process data are sampled.
P	Column vector of the gauge pressure within the furnace freeboard.
idxTrain	Logical column vector, with ones indicating observations in the training set.
minimumWarning	Number of samples preceding a blowback where a prediction would be too late.
predictionPeriod	Number of samples succeeding a blowback prediction where a prediction is valid.
Function outputs	
idxTarget	Logical column vector, with ones indicating observations in the target data.
idxFault	Logical column vector, with ones indicating where blowback-preceding conditions are present, and where warnings are early enough to be useful.
idxNormal	Logical column vector, with ones indicating observations where blowback-preceding conditions are absent.

E.3 MATLAB implementation of discriminant evaluation

The function used to calculate the discriminant rank percentile – the discriminant percentile where an observation will be flagged as faulty given the recognition window length – is given below:

```
function discriminant = discriminantPresentation(E,lw)
    % For recognition window length lw, calculate critical number of
    % threshold violations before a fault is flagged:
    r = sum(binocdf([0:lw],lw,0.75)<0.9);
    nThresh = 200;
    % Discriminant is inverse of reconstruction error:
    d = 1./E;
    % Threshold percentile where a fault is indicated, taking into account
    % moving window discriminant evaluation, is calculated below:
    discriminant = sum((movsum((repelem(d,1,nThresh)...
        -prctile(d,linspace(0,100,nThresh)))>=0,[lw,0],1)-r)>0,2);
    % The discriminant is rescaled so that it ranges between zero and 100:
    maxDisc = max(discriminant);
    discriminant = discriminant/maxDisc*100;
end
```

Function inputs	
E	Column vector of reconstruction errors.
lw	Recognition window length.
Function output	
discriminant	Column vector of discriminant rank percentiles.

E.4 MATLAB implementation of model performance evaluation

The function used to calculate the minimum specificity where 100 % sensitivity would achieve a minimum precision is given below:

```
function minSpec = calculateMinimumSpecificity(idxFault,idxNormal,minPrec)
    % 'F1' gives the total number of faulty observations:
    F1 = sum(idxFault);
    % 'F0' gives the total number of normal observations:
    F0 = sum(idxNormal);
    % Calculate the minimum specificity required for 95 % precision:
    minSpec = (minPrec-1)/minPrec*F1/F0+1;
end
```

Function inputs	
minPrec	Minimum desired precision
idxFault	Logical column vector, with ones indicating where blowback-preceding conditions are present, and where warnings are early enough to be useful.
idxNormal	Logical column vector, with ones indicating observations where blowback-preceding conditions are absent.
Function output	
minSpec	Minimum required specificity to achieve desired precision.

The function below is used to calculate the specificity and sensitivity of a sequence of discriminant values.

```
function [specificity,sensitivity] = calculateSpecificitySenstivity(discriminant,...
                                        idxFault,idxNormal)
    nThresh = 300; % Specifies the number of thresholds
    % The discriminant values where an observation is flagged is calculated
    % for each threshold value:
    d = (repelem(discriminant,1,nThresh)-linspace(0,100,nThresh))>=0;
    % The entries in a confusion matrix for each threshold value are
    % calculated:
    TP = sum(d(idxFault,:),1);   % True positives at each threshold
    TN = sum(~d(idxNormal,:),1); % True negatives at each threshold
    FP = sum(d(idxNormal,:),1);  % False positives at each threshold
    FN = sum(~d(idxFault,:),1);  % False negatives at each threshold

    specificity = TN./(TN+FP); % Specificity at each threshold
    sensitivity = TP./(TP+FN); % Sensitivity at each threshold
end
```

Function inputs	
discriminant	Column vector of discriminant rank percentiles.
idxFault	Logical column vector, with ones indicating where blowback-preceding conditions are present, and where warnings are early enough to be useful.
idxNormal	Logical column vector, with ones indicating observations where blowback-preceding conditions are absent.
Function outputs	
specificity	Row vector of specificities obtained at varying recognition thresholds.
sensitivity	Row vector of sensitivities obtained at varying recognition thresholds.

The function used to calculate the specificities and sensitivities where the minimum precision can be achieved is given below:

```
function [pSpec,pSens,pAUC] = calculatePartialAUC(specificity,sensitivity,...
                                                  minSpec)
    nThresh = 300; % Specifies the number of thresholds
    % Curve is evaluated between the minimum specificity required for 95%
    % precision and the maximum possible specificity:
    pSpec = linspace(minSpec,1,nThresh);
    % Duplicate specificities can occur at 100% specificity. 'idx' gives
    % the indices of the unique specificities.
    [~,idx] = unique(specificity);
    % The sensitivities required for 95 % precision are evaluated:
    pSens = interp1(specificity(idx),sensitivity(idx),pSpec);
    % The partial area under the curve is computed:
    pAUC = trapz(pSpec,pSens)/(1-minSpec);
end
```

Function inputs	
specificity	Row vector of specificities obtained at varying recognition thresholds.
sensitivity	Row vector of sensitivities obtained at varying recognition thresholds.
minSpec	Minimum required specificity to achieve desired precision.
Function outputs	
pSpec	Row vector of specificities obtained on the ROC-curve where the minimum precision can be achieved for varying recognition thresholds.
pSens	Row vector of sensitivities obtained on the ROC-curve where the minimum precision can be achieved for varying recognition thresholds.
pAUC	Partial area under the ROC-curve, where the minimum precision can be achieved

E.5 MATLAB implementation of PCA FPR models

The function below, 'deriveModelParametersPCA', is used to derive PCA model parameters. It corresponds to the PCA model derivation algorithm presented in Table 5.4 in section 5.5:

```
function [V,mu,sig] = deriveModelParametersPCA(X,v,l,idxTarget)
    Xl = lagmatrix(X,0:l); % Lags the entire dataset
    Xt = Xl(idxTarget,:);  % Selects target observations from lagged data
    mu = mean(Xt,1);       % Calculate target means
    sig = std(Xt,1);       % Calculate target standard deviations
    Zt = (Xt-mu)./sig;     % Standardize target data
    C = (Zt')*Zt;          % Calculate correlation matrix of target data
    % Calculate and sort principal components according to significance:
    [P,D] = eig(C);
    D = diag(D);
    [~,i] = sortrows(D,'descend');
    P = P(:,i);
    % Select retained principal components:
    V = P(:,1:v);
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
v	Integer describing the number of principal components to retain.
l	Lag dimension.
idxTarget	Logical vector indicating which samples to use as target process data.
Function outputs	
V	Matrix of retained principal components. Columns indicate different components.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.

The function below, 'calculateReconstructionErrorPCA', is used to calculate reconstruction errors using PCA model- and design parameters. It corresponds to the algorithm given in Table 5.5.

```
function E = calculateReconstructionErrorPCA(X,V,mu,sig,l)
    Xl = lagmatrix(X,0:l); % Lags the entire dataset
    Z  = (Xl-mu)./sig;     % Standardize the entire dataset
    Zr = Z*V*(V');         % Reconstruct with principal components
    E  = sum((Z-Zr).^2,2); % Calculate reconstruction error
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
V	Matrix of retained principal components. Columns indicate different components.
l	Lag dimension.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.

E.6 MATLAB implementation of kernel PCA FPR models

The function below, 'deriveModelParametersKernelPCA', is used to derive kernel PCA model parameters. It corresponds to the algorithm given in Table 5.7.

```
function [Av,Ap,Lt,K,mu,sig,kernelWidth] = deriveModelParametersKernelPCA(...
                            X,v,l,idxTarget,nCentroids)
    Xl = lagmatrix(X,0:l); % Lag the data
    Xt = Xl(idxTarget,:);  % Select target observations from lagged data
    mu = mean(Xt,1);       % Calculate target data means
    sig = std(Xt,1);       % Calculate target data standard deviations
    Zt = (Xt-mu)./sig;     % Standardize target data
    % Use k-means clustering to obtain centroid dataset:
    Lt = kmeans(Zt,nCentroids,'MaxIter',1000,'Replicates',100,...
                    'Distance','sqeuclidean');
    % Calculate squared distance matrix of centroid dataset:
    Dt = pdist(Lt); Dt = squareform(Dt).^2;
    % Calculate optimal kernel width:
    kernelWidth = fmincon(@(x) findOptimalWidth(x,Dt,v),5000,[],[],200,10000);
    % Calculate kernel matrix of target data:
    K = exp(-Dt/kernelWidth); K(K<=e-4)=0;
    % Center the kernel matrix of target data:
```

```
    U = ones(nCentroids,nCentroids)/nCentroids;
    Kc = K - U*K - K*U + U*K*U;
    % Calculate the principal linear combinations of the kernel matrix:
    [A,D] = eig(Kc); D = diag(D); [D,i] = sortrows(D,'descend');
    % Sort linear combinations according to principal component
    % significance:
    A = A(:,i);
    % Standardize linear combinations:
    A = A./((D.^0.5)');
    % Find the number of non-zero eigenvalues:
    n = find(D<1e-5,1)-1;
    % Calculate the retained principal linear combinations as well as the
    % non-zero principal linear combinations:
    Av = A(:,1:v);
    Ap = A(:,1:n);
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
v	Integer describing the number of principal components to retain.
l	Lag dimension.
idxTarget	Logical vector indicating which samples to use as target process data.
nCentroids	Number of clusters to use to approximate the target data.
Function outputs	
Av	Matrix of retained principal linear combinations.
Ap	Matrix of non-zero principal linear combinations.
Lt	Matrix of centroids obtained through k-means clustering of the target data.
K	Kernel matrix.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.
kernelWidth	Optimal kernel width.

The function below, 'calculateReconstructionErrorKernelPCA', is used to calculate reconstruction error using kernel PCA design- and model parameters. It corresponds to the algorithm given in Table 5.8.

```
function E = calculateReconstructionErrorKernelPCA(X,Av,Ap,K,mu,sig,...
                                        kernelWidth,nCentroids,l,Lt)
    Xl = lagmatrix(X,0:l); % Lag the data
    Z  = (Xl-mu)./sig;     % Standardize lagged data with target means- and
                           % standard deviations
    % Calculate the squared distance between each observation and the
    % centroid observations:
    D1 = (pdist2(Lt,Z)').^2;
    % Calculate the kernel matrix of the new observations:
    K1 = exp(-D1/kernelWidth);
    % Center the new kernel matrix:
    N1 = size(K1,1);
    U  = ones(nCentroids,nCentroids)/nCentroids;
    u  = ones(N1,nCentroids)/nCentroids;
    K1c = K1 - u*K - K1*U + u*K*U;
    % Calculate the projection of the new observations on the retained- and
    % non-zero principal linear combinations:
```

```
    tv = K1c*Av;
    tp = K1c*Ap;
    % Calculate the reconstruction error:
    E  = sum(tp.*tp,2)-sum(tv.*tv,2);
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
Av	Matrix of retained principal linear combinations.
Ap	Matrix of non-zero principal linear combinations.
Lt	Matrix of centroids obtained through k-means clustering of the target data.
l	Lag dimension.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.
kernelWidth	Optimal kernel width.
nCentroids	Number of clusters to use to approximate the target data.

The objective function that is minimized to obtain an optimal kernel width is defined below:

```
function kernelWidthObjective = findOptimalWidth(x,Dt,v)
    K = exp(-Dt/x); % Calculate the kernel matrix
    % Center the kernel matrix:
    nl = size(K,1);
    U = ones(nl,nl)/nl;
    Kc = K - U*K - K*U + U*K*U;
    % Calculate the retained variance on each principal component in the
    % nonlinear feature space:
    [~,D] = eig(Kc);
    % Sort the retained variance:
    D = diag(D); D = sortrows(D,'descend');
    D = D(D>=1e-5);
    % The objective is the negative of the fraction of variance on the
    % retained components:
    kernelWidthObjective = -sum(D(1:v))/sum(D);
end
```

Function inputs	
x	Kernel width, varied to minimize the objective value.
Dt	Squared distance matrix of the centroid dataset.
v	Integer describing the number of principal components to retain.
Function outputs	
kernelWidthObjective	Value that is minimized to maximize variance on retained components.

E.7 MATLAB implementation of auto-encoder FPR models

The algorithm used to derive model parameters for an auto-encoder is given in the function, 'deriveModelParametersAutoEncoder', below. It corresponds to the algorithm given in Table 5.10.

```
function [net,mu,sig] = deriveModelParametersAutoEncoder(X,l,nHidden,c,...
                                                         idxTarget)
    Xl = lagmatrix(X,0:l); % Lag the data
```

```
    Xt = Xl(idxTarget,:);  % Select target observations in lagged data
    mu = mean(Xt,1);       % Calculate target means
    sig = std(Xt,1);       % Calculate target standard deviations
    Zt = (Xt-mu)./sig;     % Standardize the target observations
    % Corrupt the standardized target observations:
    m = size(Zt,2); n = size(Zt,1);
    Ztc = Zt + normrnd(0,c,n,m);
    % Set up the auto-encoder neural network:
    net = fitnet([m*2,nHidden,m*2]); % Specify network architecture
    net.trainFcn = 'traingdm'; % Gradient descent with momentum
    net.divideFcn = 'dividetrain'; % All observations are used for training
    net.trainParam.epochs = 2000; % Number of training epochs
    net.performFcn = 'mse'; % Loss function to be minimized.
    net.layers{1}.transferFcn = 'poslin'; % ReLU activation for first layer
    net.layers{2}.transferFcn = 'poslin'; % ReLU activation for second layer
    net.layers{3}.transferFcn = 'poslin'; % ReLU activation for third layer
    net.trainParam.showWindow = 0; % Do not display training, speeds up training
    net.performParam.regularization = 1e-5; % L2 regularization constant
    Ztc = Ztc'; Zt = Zt'; % Switch rows and columns for training network
    net = train(net,Ztc,Zt); % Optimize network parameters
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
nHidden	Integer of the number of neurons in the auto-encoder hidden layer.
c	Variance of the noise used to corrupt target data before training the network.
l	Lag dimension.
idxTarget	Logical vector indicating which samples to use as target process data.
Function outputs	
net	Trained auto-encoder neural network.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.

The algorithm used to calculate reconstruction error using a derived auto-encoder is defined below, it corresponds to the algorithm given in Table 5.11:

```
function E = calculateReconstructionErrorAutoEncoder(X,mu,sig,l,net)
    Xl = lagmatrix(X,0:l); % Lag the data
    Z  = (Xl-mu)./sig; % Standardize the lagged data
    Z  = Z'; % Switch rows and columns for network
    Zr = net(Z); % Reconstruct data
    Z  = Z'; % Switch rows and columns back for original data
    Zr = Zr'; % Switch rows and columns for reconstructed data
    E = sum((Z-Zr).^2,2); % Calculate reconstruction error
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.
l	Lag dimension.

net	Trained auto-encoder neural network.

E.8 MATLAB implementation of convolutional auto-encoder FPR models

The algorithm used for deriving a convolutional auto-encoder's model parameters is given in the function below:

```
function [cNet,mu,sig] = deriveModelParametersConvolutionalAE(X,...
                                    networkType,c,idxTarget)
    mu  = mean(X(idxTarget,:),1); % Calculate target data means
    sig = std(X(idxTarget,:),1);  % Calculate target data standard deviations
    Z   = (X-mu)./sig;      % Standardize the data
    n = size(Z,1); m = size(Z,2); % Find data dimensions
    switch networkType
        case 'short'
            l = 4; % Lag dimension for short architecture
            Z = lagmatrix(Z,0:l); % Lag the data
            % Convert data to sequence of MTS-images:
            Z = reshape(reshape(reshape(Z',m,n*(l+1)),m,l+1,n),m,l+1,1,n);
            % Select target data from images:
            Zt = Z(:,:,1,idxTarget); nt = size(Zt,4);
            % Create corrupted target data:
            Ztc = Z + normrnd(0,c,m,l+1,nt);
            % Specify short CAE architecture:
            layers = [...
                imageInputLayer([9,5,1])
                convolution2dLayer([1,3],8)
                reluLayer
                convolution2dLayer([1,3],8)
                reluLayer
                convolution2dLayer([9,1],4)
                reluLayer
                transposedConv2dLayer([9,5],1)
                regressionLayer];
        case 'long'
            l = 6; % Lag dimension for long architecture
            Z = lagmatrix(Z,0:l); % Lag the data
            % Convert data to sequence of MTS images:
            Z = reshape(reshape(reshape(Z',m,n*(l+1)),m,l+1,n),m,l+1,1,n);
            % Select target data:
            Zt = Z(:,:,1,idxTarget); nt = size(Zt,4);
            % Create corrupted target data:
            Ztc = Z + normrnd(0,c,m,l+1,nt);
            % Specify long CAE architecture:
            layers = [...
                imageInputLayer([9,7,1])
                convolution2dLayer([1,3],8)
                reluLayer
                convolution2dLayer([1,3],8)
                reluLayer
                convolution2dLayer([1,3],8)
                reluLayer
                convolution2dLayer([9,1],4)
                reluLayer
                transposedConv2dLayer([9,7],1)
                regressionLayer];
```

```
    end
    % Specify training options:
    options = trainingOptions('sgdm', ... % Stochastic gradient descent with momentum
    'InitialLearnRate',0.0001, ... % Initial learning rate
    'Verbose',false, ... % Do not display training information
    'Plots','none', ... % Do not plot progress
    'MaxEpochs',100, ... % Maximum number of training epochs
    'L2Regularization',1e-2); % L2 regularization constant employed
    cNet = trainNetwork(Ztc,Zt,layers,options); % Train CAE network
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
networkType	Specifies the convolutional auto-encoder architecture
c	Variance of the noise used to corrupt target data before training the network.
idxTarget	Logical vector indicating which samples to use as target process data.
Function outputs	
cNet	Trained convolutional auto-encoder neural network.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.

The following algorithm is used to calculate reconstruction error using a derived convolutional auto-encoder:

```
function E = calculateReconstructionErrorConvolutionalAE(X,cNet,mu,sig,networkType)
    Z = (X-mu)./sig; % Standardize data
    % Convert data to sequence of MTS-images according to network type:
    n = size(Z,1); m = size(Z,2);
    switch networkType
        case 'short'
            l = 4;
            Z = lagmatrix(Z,0:l);
            Z = reshape(reshape(reshape(Z',m,n*(l+1)),m,l+1,n),m,l+1,1,n);
        case 'long'
            l = 6;
            Z = lagmatrix(Z,0:l);
            Z = reshape(reshape(reshape(Z',m,n*(l+1)),m,l+1,n),m,l+1,1,n);
    end
    Ei = sse(Z,predict(cNet,Z)); % Calculate reconstruction error
    E  = double(Ei(:)); % Convert error to double vector-array
end
```

Function inputs	
X	Matrix of monitored variables, different variables are in different columns and different samples are in different rows.
mu	Row-vector of target data means.
sig	Row-vector of target data standard deviations.
networkType	Specifies the convolutional auto-encoder architecture
cNet	Trained convolutional auto-encoder neural network.